SDN-enabled Workload Offloading Schemes for IoT Video Analytics Applications

Pouria Pourrashidi Shahrbabaki

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

September 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Pouria Pourrashidi Shahrbabaki**

Entitled: **SDN-enabled Workload Offloading Schemes for IoT Video Analytics Applications**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. M. Reza Soleymani

_____ Examiner
Dr. Sandra Cespedes

_____ Examiner
Dr. M. Reza Soleymani

_____ Supervisor
Dr. Yousef R. Shayan

_____ Supervisor
Dr. Rodolfo W. L. Coutinho

Approved by _____
Dr. Yousef R. Shayan, Chair
Department of Electrical and Computer Engineering

_____ 2023

_____
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

SDN-enabled Workload Offloading Schemes for IoT Video Analytics Applications

Pouria Pourrashidi Shahrbabaki

Increasing demand for using IoT applications, such as video analytics, leverages the importance of developing an architecture to meet the requirements in terms of the latency, reliability, and energy consumption. IoT video cameras combined with the power of machine learning algorithms introduce real-time video analytics applications that can be used in diverse domains, such as security surveillance, sports, and retail stores. However, processing captured video frames using machine learning algorithms needs resources that are beyond the capability of these IoT devices.

IoT task offloading is a new paradigm to aim IoT applications to deliver processing intensive applications to their users. IoT devices, which have limited resources by nature, offload their tasks to more powerful servers, i.e., edge/cloud servers . Nonetheless, selecting an appropriate destination for offloading the tasks is the first incoming problem for the IoT task offloading. There are some criteria which needs to be considered when it comes to IoT task offloading, for example transmission latency, queuing delay, as well as processing latency. Although edge servers have limited resources compared to cloud servers, the end-to-end latency for sending the packets to the edge servers is less than the cloud servers. On the other hand, because of the limited available resources in the edge servers, distributing the offloaded tasks between these devices is necessary to avoid overloaded servers.

Considering the above mentioned facts, in this thesis, we present load-balancing algorithms benefits from Software Defined Networking (SDN) to distribute offloaded tasks to reduce the chance of using overloaded servers and processing latency of offloaded packets of IoT video analytics applications. Taking into account the aforementioned facts, we propose a scoring metric to balance the incoming offloaded packets between edge servers. The introduced algorithm takes advantage of

underlying SDN to collect information about the load of each edge server in the network. Then, the SDN controller uses the scoring metric and sorts the edge servers accordingly. The offloaded task will be directed to the edge server with the lowest processing load to avoid overloaded edge servers.

Since the number of IoT devices in the network is not predictable, increasing number of IoT devices will lead to overloaded edge servers. Hence, offloading a part of the IoT tasks to the cloud server might be a better option, even though the packets should pass through the core network. In this regard, we developed a hierarchical edge/cloud system for IoT task offloading. We modeled each of edge/cloud servers by M/M/1 queue model. By benefiting from SDN as an underlying network, the SDN calculates the processing latency and transmission latency to edge and cloud servers, and decides the best destination in terms of the minimum latency that directs the offloaded tasks to one of the desired servers.

We have conducted extensive performance evaluation to demonstrate the out-performance of the developed solutions compared with other related approaches in terms of total experienced latency and load distribution between the available servers. The results are comprehensively discussed in their related chapters to clarify the performance of the developed solution.

Acknowledgments

I would like to express my deepest gratitude to my supervisors, Prof. Yousef R. Shayan and Prof. Rodolfo W. L. Coutinho for providing me this great opportunity and leading the way to grow myself not only as a student but also as a person.

I also would like to thank my parents, sister, family, and friends for all of their support and love they gave me from beginning of this path to the end. It wouldn't be possible to reach to this point without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's world, IoT applications play a vital rule in daily human life. Different IoT applications in diverse categories can be observed. IoT applications can be used in areas such as environment monitoring and security surveillance, health care, smart industries, as well as agriculture (Khanna and Kaur (2020)). IoT video analytics is also one of the IoT applications which has brought a lot of attraction. Deploying several IoT cameras to capture videos can be used to detect object of interest from the streamed video frames. Deep learning algorithms can be used to process video frames in a wide range of applications, such as detecting defected products in an industrial floor or managing traffic in big cities (Coutinho and Boukerche (2022b); X. Liu et al. (2020)). However, resource-bare IoT devices are not able to process these streamed video frames with high quality to detect objects of interest. Although some IoT cameras might be able to process poor quality video frames, decreasing the quality of streamed videos results in narrowing the outcome of the deep learning algorithms to detect objects.

To deal with the mentioned problem, some articles have proposed solutions to offload streamed video frames to cloud servers (Altamimi et al. (2015); Hu and Xiao (2021)). However, delay of the core network is not tolerable for latency sensitive character of video analytics applications. Therefore, edge computing paradigm has been introduced to bring computing resources to the edge of the network to avoid high latency of transmission to the cloud servers. In fact, edge computing is a distributed computing paradigm that brings processing resources close to the clients as can be seen in the Fig.1.1 (Varghese et al. (2016)).

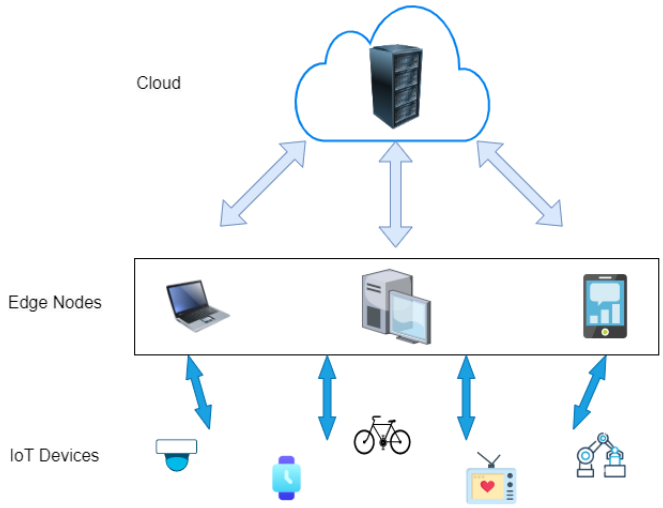


Figure 1.1: Edge Computing Paradigm

1.1 Thesis Motivation

Despite the fact that edge nodes bring benefits of distributed computing to the IoT applications, and can reduce the transmission delay to offload the task from IoT devices to edge nodes, these edge servers have limited resources compared to cloud servers. Therefore, if the offloaded tasks from the IoT devices to an edge server are more than the processing capacity of that server, we face overload in the edge server. Hence, in the edge servers, offloaded packets will face high queuing delay and chance of packet drop, which as a result will increase the delay of task offloading. In another word, the incoming flows from the IoT devices should be managed in such a way that the incoming load is balanced between the edge and cloud servers as well as avoiding intensive task offloading to the edge servers based on their available resources to stay away from overloaded servers.

Considering the above-mentioned facts, one of the problems is how to use these limited resources of edge servers effectively, in addition to, considering the capabilities of cloud servers to meet the ever-evolving requirements of IoT networks, with their goals, demands, and user population constantly changing. Hence, an offloading policy is needed to manage incoming flows taking into account the edge and cloud servers as well as network loads. However, one of the main challenges is to receive the load of the network and edge as well as cloud servers. In addition, a fast and scalable architecture is mandatory to implement the proposed offloading policy. In this regard, there

are some articles in the literature, which have proposed paradigms to address the mentioned challenges. For example, P. Li et al. (2022) addressed the problem that uneven distribution of Internet of Vehicles (IoVs) and resources in edge nodes leads to uneven load balancing in the edge servers. Therefore, Deep Q-Network (DQN) model is proposed by the authors using SDN networks. They introduced a three layers hierarchical architecture, in which the top layer is Master SDN Controller (MSDNC) that is in charge of managing the whole network. The second layer is RSU, Edge Server, and Edge SDN Controller (ESDNC), and the third layer includes the end users. In their model when a user tries to offload its task to a nearby edge server, if the load at the edge server is less than its load threshold, the edge server will accept the offloading. However, if the load at the edge server is higher than the threshold, the task offloading would be accepted if the task can't be done by the user. When the load at edge server is higher than its threshold, the ESDNC will inform the MSDNC for load balancing procedure. MSDNC by using Q-learning and Deep Neural Network performs the load balancing policy to offload the task from overloaded edge servers to underloaded ones.

J. Li et al. (2020a) proposed a task offloading framework for Mobile Edge Computing (MEC) in IoT applications. The proposed framework aims at finding an optimal offloading decision to improve the quality of experience (QoE) of IoT devices and minimize transmission delay and energy consumption. The proposed framework estimates the task's time-to-live (TTL) and compares it with the task completion time when processed locally and in the edge server. The offloading decision mechanism offloads the task to the edge server when the task's TTL is greater than its completion time in the edge server and less than the completion in IoT device. If the task's TTL is greater than the completion time of the task in the edge server and the IoT device, the scheme will compare the computation cost of the IoT device and edge server. If the edge server has a lower computation cost, the task will be offloaded.

Y. Chen et al. (2017) proposed a joint optimization algorithm to reduce transmission as well as processing delay. The authors modeled each fog server as M/M/1 queue model. Their optimization algorithm is subject to that each user equipment is assigned to one fog server, the processing delay should be less than an assigned time, and transmission delay from fog node to user equipment should be greater than zero. The authors used Simulated Annealing (SA) algorithm to solve their optimization problem.

The above articles introduced algorithms for task offloading to distribute the tasks between the edge servers. However, for dense networks some articles introduce cloud servers into their architecture. In their proposed systems, the authors considered the transmission latency of the WAN to make proper task offloading decisions between edge and cloud servers. These articles are addressed in the following paragraphs.

Deng et al. (2016) addressed the high power consumption in cloud servers due to increasing resource demands as well as the applications' delay limitation. In their proposed framework, the authors formulated the fog and cloud processing delay, transmission delay from fog to cloud server, as well as fog and cloud server power consumption. Then, they introduced an optimization problem to minimize the total power consumption of the system. In order to solve the optimization problem, the authors divided the main challenge into three sub-problems. Then, by using interior-point methods, generalized Benders decomposition (GBD), and Hungarian method, they solved the sub-problems, and approximately reached a final answer for the main optimization problem.

Kai et al. (2021) developed a hierarchical architecture with mobile devices, edge servers, and cloud servers. The authors modeled the communication delay between a mobile device and its designated edge server as well as from the edge server to the cloud server. In addition, the authors modeled computation delay in each of the mobile devices, edge servers, and cloud server. The authors introduced a pipeline strategy aiming at reducing the total computation delay of mobile devices' tasks where at first a mobile device will decide what percent of its tasks should be offloaded, then the appointed edge node will offload the tasks which are beyond its capacity to the cloud server.

The referred works introduced solutions to improve Quality of Service (QoS) in IoT applications. In their developed systems, their main objective is to maintain the processing delay below a certain time. However, load balancing is the missing part of their solutions. Considering available resources, i.e., edge and cloud servers in a jointly-manner is a crucial part in IoT task offloading. Besides, having a mechanism to capture the load at each server is a vital requisition to be able to make a decent decision to select an appropriate server for incoming flows. Therefore, in this thesis, our solutions divide the incoming load between edge and cloud servers taking into account their current load aiming at minimizing the total experienced delay.

1.2 Thesis Contribution

Considering the aforementioned facts, in this thesis, we proposed load balancing algorithms for IoT task offloading to reduce the processing latency described in the two following contributions:

1.2.1 A Novel SDN-enabled Edge Computing Load Balancing Scheme for IoT Video Analytics:

We have proposed an algorithm to distribute the offloaded streamed video frames between edge servers, using SDN to orchestrate the incoming flows for scenarios where delay of WAN is very high or the available resources at the edge is sufficient to meet the requirements of the IoT cameras. In this regards:

- We have proposed a model for IoT video analytics task offloading to calculate the latency to process the offloaded streamed videos from IoT devices to edge servers.
- A scoring metric is proposed to calculate the load at each edge server considering the incoming and outgoing traffic. All the required information to calculate scoring metric are gathered by SDN controller from OpenFlow switches under its domain. Based on the scoring metric, the incoming traffic from a real-time video analytics application will be offloaded to an edge server with minimum load.
- We designed an algorithm to use all the mentioned information in the SDN controller to find an edge server with the minimum workload. The proposed solution aims at minimizing the processing latency and prevents overloaded edge servers.
- Mininet network emulator and POX SDN controller have been used to assess and indicate the superiority of the designed algorithm.

1.2.2 SDN-LB: A Novel Server Workload Balancing Algorithm for IoT Video Analytics:

In dense networks with limited resources at the edge, offloading part of IoT cameras' video frames to cloud servers might come to the aid of time-sensitive applications to decrease the total

latency and the load on the edge servers. With respect to that:

- We considered a hierarchical edge/cloud servers architecture, where each of these servers are modeled by M/M/1 queuing computation.
- The proposed queuing model calculates average processing delay considering computation resources in the servers and streamed video frames configuration.
- SDN as a brain of the designed architecture gathers information about edge and cloud servers to direct the incoming flows between the servers by using the designed SDN-LB algorithm based on the transmission latency at the core or LAN network and processing latency at the servers.
- To demonstrate the capability of the proposed contribution, the developed solution has been evaluated and compared with related algorithms for IoT task offloading under extensive set of numerical evaluations by Python programming language.

1.3 Thesis Outline

The rest of the thesis is organized as follows:

- **Chapter 2:** In this chapter we present background and prior related works relevant to the thesis. We discuss in more detail the concepts of edge and cloud computing.
- **Chapter 3:** In this chapter we developed a novel SDN-enabled edge computing load balancing scheme for IoT video analytics.
- **Chapter 4:** This chapter introduces an algorithm named SDN-LB: a novel server workload balancing algorithm for IoT video analytics,
- **Chapter 5:** This chapter concludes the thesis and addresses the future works.

Chapter 2

Background and Related Work

In this chapter, the fundamental building blocks, which are required for understanding the thesis are explained.

2.1 IoT Video Analytics

In the realm of IoT video analytics, the challenge of identifying objects of interest, in scenarios such as traffic management and surveillance, in dynamic environments such as big and manipulated cities has become a pivotal concern in the modern world. To address this challenge, usually thousands of cameras are deployed across these urban landscapes. However, one of the challenging problems is how to analyse the streamed video frames in a real-time manner. One immediate approach has been to employ human analysts to analyse the streamed videos. Nonetheless, this solution seems to be impractical as the number of IoT cameras increases. Besides, human nature is error prone, which can have significant consequences in critical applications.

Recognizing the limitations of human analysis and the need for more efficient solutions, machine learning solutions are devoted as a promising avenue to analyze massive amounts of streaming video data. Machine learning algorithms, such as Convolutional Neural Networks (CNNs), can process video frames for various purposes, including facial recognition for enhanced security measures, analyzing customer behavior in retail settings to identify potential customers, detecting items within warehouses to pinpoint defective products, and sport events (J. Chen and Jenkins (2017),

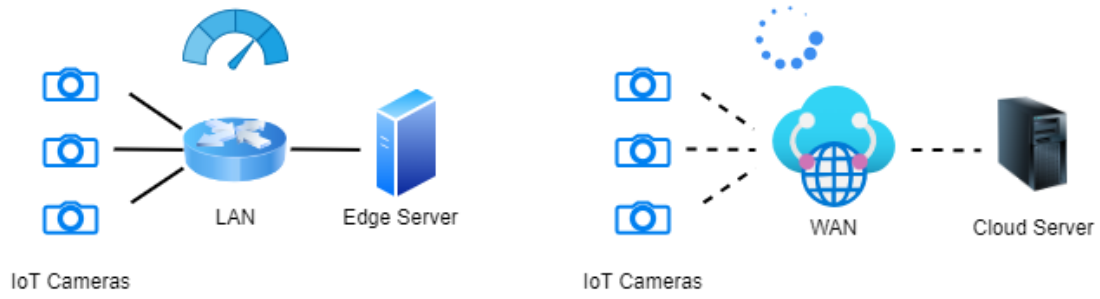


Figure 2.1: Edge vs Cloud servers

Nogueira et al. (2019), Coutinho and Boukerche (2022b)). Nevertheless, the processing demands of complex machine learning algorithms often surpass the capabilities of these resource-constrained IoT devices. This brings into focus the crucial issue of resource allocation and optimization, as these devices grapple with the computational demands of real-time video analysis.

2.2 Edge Computing

Edge computing is a distributed information technology architecture, designed to optimize and enhance data processing by positioning computing resources in close proximity to end-users. This strategic positioning serves a vital purpose: minimizing transmission delays that can occur when data needs to traverse great distances to reach traditional data centers as can be observed in Fig.2.1. Increasing data producing of billions IoT devices and sensors, bandwidth limitation, and uncertainly characteristics of the core network are some of the reasons that make it impossible to send this amount of data through the WAN to central data servers which are in the cloud.

Furthermore, the dynamics of latency-sensitive applications underscore the limitations of traditional central server models. Applications that demand real-time responsiveness, such as remote robotic control, augmented reality, and autonomous systems (Alfakih et al. (2020); Coutinho and Boukerche (2019)), cannot afford the transmission latency incurred by routing data to distant servers and waiting for the response, since the transmission latency to send the data to distant cloud servers can be up to 60 times more compared to the edge servers Goudarzi et al. (2021). The demand for low-latency communication has lead the development of edge computing, which directly addresses

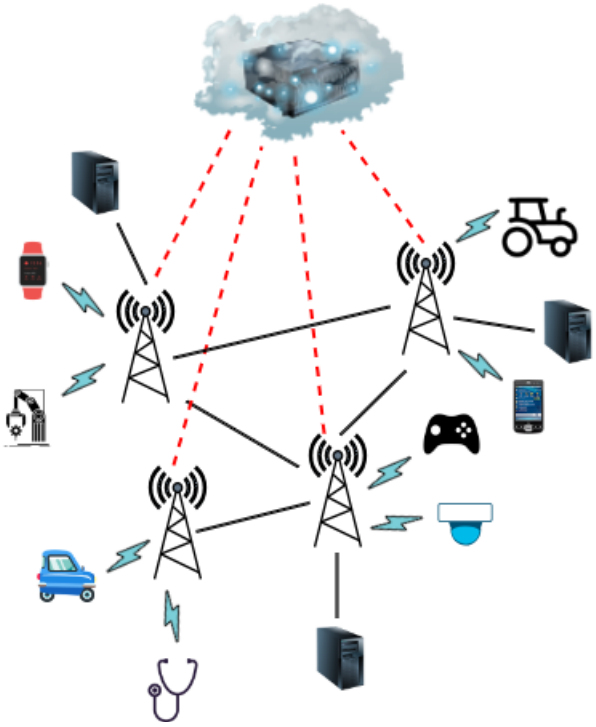


Figure 2.2: Edge-Cloud computing system

this problem by placing computational and storage resources at the network edge, in proximity to where data originates. By co-locating computing resources with the IoT devices and sensors which generates data, edge computing minimizes the physical distance that data must travel, resulting in reduced latency, where Fig.2.2 points to it.

To benefit from edge computing in the IoT networks, some research has been done, and articles proposed different architectures. For example, Sun and Ansari (2016) introduced an architecture for IoT edge computing, named EdgeIoT. In their developed architecture, edge nodes are connected to Base Stations or deployed at the edge of the cellular core network. Also, these nodes are connected to the cloud servers through the Internet in case that they did not have enough resources to process the offloading tasks. In their architecture, each user has been assigned a proxy VM in the nearby node. First, the user's data offloads to proxy server, then the proxy server groups the user data into several categories, and later it offloads each group to its appropriate node.

Tong et al. (2016) developed a hierarchical architecture to reduce task processing delay and also avoid overloaded edge servers. In the proposed architecture the servers are categorized based

on their resources. Edge servers with higher computation capacity are located in the higher tiers, however, their distance from users increase. The article aims at distributing offloaded tasks among the servers, taking into account the transmission delay as well as workload of each tier servers.

Xu et al. (2019) mentioned the problems with the arbitrary IoT task offloading, and disabilities to reduce energy consumption and computation latency. To solve the aforementioned problem, the authors developed a paradigm, named COM. In COM, each IoT application has been considered as a directed acyclic graph, which is divided into multiple tasks. Proposed paradigm considers network latency of task offloading, transmission delay to exchange data between tasks, and energy consumption. The authors used non-dominated sorting genetic algorithm III to solve the optimization problem. COM is evaluated compared to Benchmark, Cloudlet-oriented Computation Offloading Method, and Cloud-oriented Computation Offloading Method.

Y. Chen et al. (2023) developed an algorithm to minimize both energy consumption of IoT devices and processing latency for offloaded tasks. In order to reach this goal the authors formulated the problem as a Markov Decision Process, and by benefiting from Deep Reinforcement Learning (DRL), they proposed an algorithm named DRL-based energy efficient task offloading (DEETO) to solve the optimization problem. The agents of DEETO are deployed in each IoT device, which decide about task offloading and a desired edge server. Agents of the DRL algorithm need information about the entities in network to make offloading decision. Therefore, the authors considered using digital twin to build virtual model of devices.

The above mentioned articles proposed architectures for IoT edge computing. However, the main concern about edge computing is that each edge server has limited resources compared to the cloud servers. Therefore, the offloaded tasks should be managed in such a way to avoid overloaded edge servers which results in increasing processing delay. Hence, an offloading policy is needed to balance the offloaded tasks between edge servers. In order to introduce an efficient offloading policy, the load balancing algorithm should be aware of the load in each edge server. In addition, leaving the offloading decision algorithms to bare-resources devices, i.e., IoT devices, will impose extra load to these battery dependent devices.

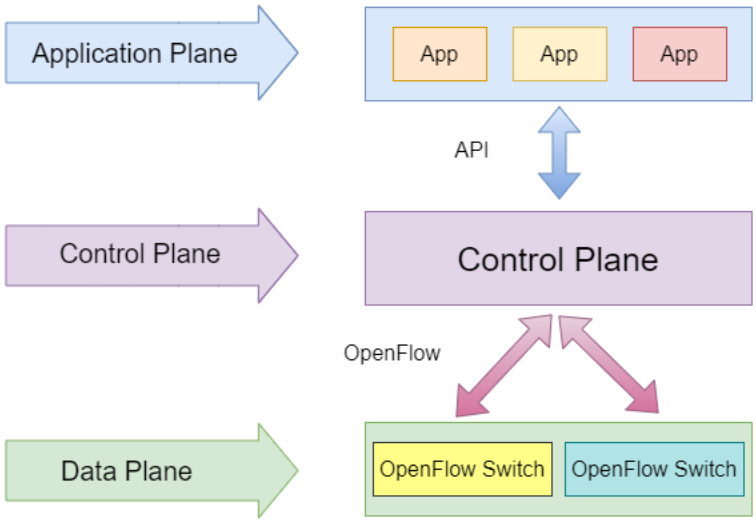


Figure 2.3: Software Defined Networking Architecture

2.3 Software Defined Networking

Software Defined Networking (SDN) suggests a new networking methodology that represents a new way that networks are managed and designed. It plans the network infrastructure in a way that control and decision-making functions (Control Plane) are separated from the data forwarding functions (Data Plane) as can be seen in Fig.2.3. SDN provides network operations that enjoy a new level of flexibility and efficiency.

SDN could be considered as an innovation that transforms the way networks are designed, managed, and operated, paving the way for a more dynamic and efficient digital landscape. Within the SDN framework, the responsibility of managing the entire network is assigned to a programmable layer called the Control Plane. This architecture brings numerous benefits in terms of reducing the complexity of traditional network's management and maintenance. In contrast to the traditional network management that needs network administrators to configure and manage each network devices from variety of vendors, the SDN controller itself deals with all these difficulties (Benzekki et al. (2016)) as we can see in Fig.2.4. In addition, SDN outperforms traditional networking in various critical aspects:

- Agility and Dynamic Management: SDN gives network managers the power to quickly adjust

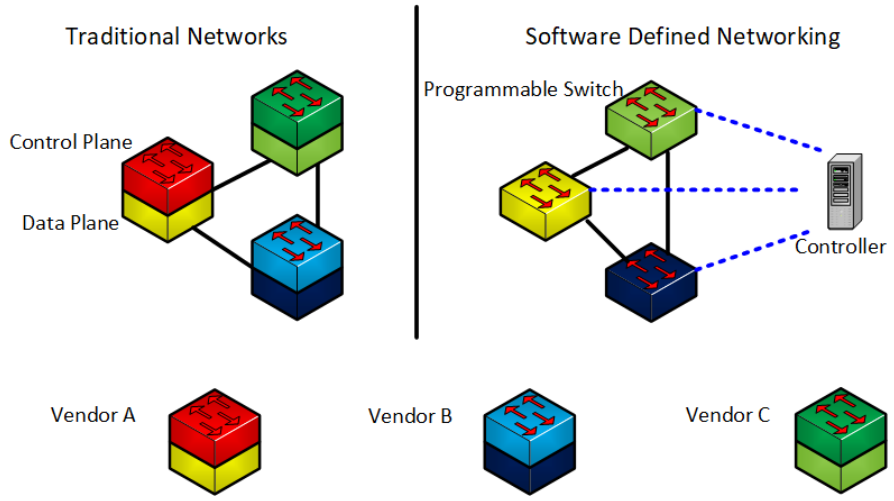


Figure 2.4: Traditional networks vs SDN

and customize network's diverse services to match changing needs by seperating the control and data plane. This ability comes from the flexible central Control Plane, which lets changes happen fast without requiring to redo the setup of each separate device on the network.

- Heterogeneous Scalability: SDN's architecture facilitates scalability based on the specific needs of network services. The decoupled nature of control and data planes of SDN enables the addition or modification of network resources without undermining the overall network efficiency.

- Logically Centralized Management: The SDN controller acts as a single point of control with full vision across the entire network. This perspective empowers administrators to make informed decisions, optimize traffic flows, and implement network-wide policies more effectively.

Although SDN has lots of advantages, there are some drawbacks that are worth to mentioning:

- SDN can lead to decrease costs in the long run. However, the initial deployment costs can be high and organizations need to invest in new infrastructure, software, and training for their IT staff.

- Migration from a traditional network to an SDN solution can be a challenging process. Integration with existing networking hardware and protocols may cause compatibility complexities.

- A centralized controller is used by SDN that can become a single point of failure. The network's security can be at risk if the controller is compromised. In addition, separating control and data planes could possibly open up new attack vectors if not properly secured.

- Any failure in the controller can lead to disruptions across the network, because SDN and network availability rely heavily on the central controller.

There exist multiple commercial and open source controllers, catering to different network environments and demands. Notable examples include NOX/POX, Ryu Controller, and Cisco's proprietary solutions. However, for SDN to effectively operate within multi-vendor environments and to enable the scaling of SDN networks, a standardized communication protocol is indispensable. OpenFlow is a protocol to communicate between SDN switches and controller. The basic concepts of the SDN were introduced in 1996, however, the development of the OpenFlow in 2011 brought many attention to SDN. There are two main components in devices that support OpenFlow: an API, which supports the communication between switches and the SDN controller, and a flow table to determine actions needed for each incoming packet (Benzekki et al. (2016)). OpenFlow acts as the language between the network's controller and the forwarding switches.

Having a controller with full vision of the underlying network can benefit in different aspects. Controller as the brain of the network, which combines with programmable switches as well as OpenFlow protocol, altogether, aiming to gain information about entities which exist in the network, i.e., IoT cameras, edge servers, and cloud servers. SDN controller can collect information such as number of transmitted and received bytes, dropped packets, as well as transmitted and received packet at each port of OpenFlow switches. By sending port status request from the SDN controller, the OpenFlow switches will respond with an event type packet with all the requested information. This request and response is a lightweight process as each request packet is 80 bytes and response is 112 bytes.

With respect to the above mentioned facts, these SDN features can give information which can

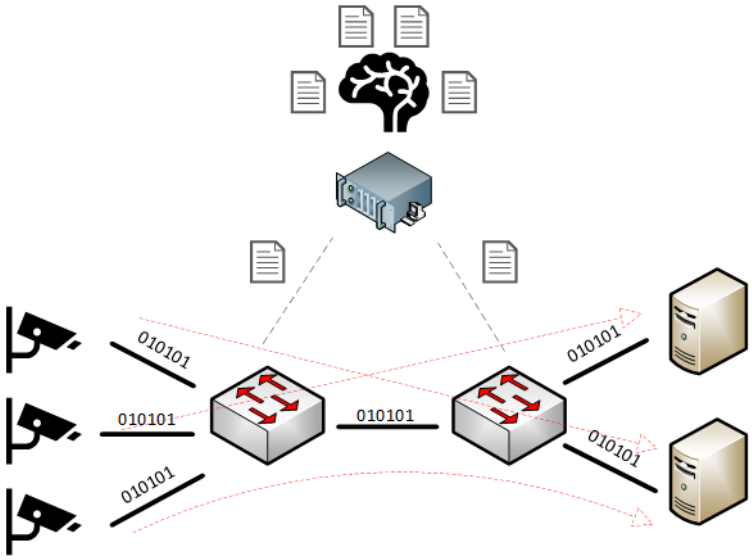


Figure 2.5: Full vision of SDN controller on the network entities

make a crucial impact on how IoT task offloading can be done. A programmable controller that can collect the required information and direct the incoming flows plays a vital rule to direct and distribute the tasks between the servers by considering their current workload. The load at each server can be calculated by the information that controller can collect from each port of OpenFlow switches as we can see in Fig.2.5. These are the reasons why we use SDN in our proposed solutions.

Chapter 3

A Novel SDN-enabled Edge Computing Load Balancing Scheme for IoT Video Analytics

Edge computing has been designed to deploy resources in the proximity of IoT devices, which reduces latency and network overhead. Nevertheless, resources on edge servers are limited and must efficiently be managed. In this chapter, we propose a novel software-defined networking (SDN)-based scheme to balance the computation resource requests among a network of edge servers aimed at supporting IoT video analytics streaming applications. In the proposed solution, programmable switches periodically report the IoT video streaming workload forwarded to each edge server. This information is then used at the SDN controller to estimate the incoming and outgoing traffic load at edge servers and balance IoT video streaming among them, by updating routing tables at the programmable switches. The performance of the proposed solution is evaluated and compared to related schemes through extensive simulations using the Mininet emulator. Obtained results show that the proposed solution can reduce up to 21 % of average latency with 20 % load saving in each edge server, compared to deterministic and random-based related solutions.

3.1 Introduction

Internet of things (IoT) has ushered smart applications in several domains, such as transportation, health care, education, smart and green buildings, entertainment, and industrial (Coutinho & Boukerche, 2020; C. Lin et al., 2020; Y. Zhang et al., 2021). IoT applications often rely on massive deployments of heterogeneous and resource-constrained devices that produce a large amount of data. Traditionally, resource-constrained IoT devices offload to cloud servers their produced data and computation-intensive tasks. Hence, cloud servers process data and computation-intensive tasks produced by IoT devices. However, the offloading process incurs additional latency for the IoT application and high overhead for the network core, which is impractical for time-sensitive applications, such as autonomous cars, industry 4.0, telesurgery, virtual/augmented/mixed reality (VR/AR/MR) reality (Coutinho & Boukerche, 2019; Holland et al., 2019; Qian & Coutinho, 2021).

In this context, the multi-access edge computing (MEC) paradigm has been proposed to deploy computation, communication, and storage resources in the proximity of IoT devices. In the MEC paradigm, edge servers are deployed at different locations in the access network to provide resources to IoT devices. Thus, IoT data and tasks are offloaded to edge servers instead of to cloud servers, which reduces latency and network overhead. Nevertheless, edge servers have limited resources when compared to cloud servers, which might degrade the performance of IoT applications. For instance, in an IoT video analytics application, the latency to process and detect objects of interest in video frames, at an edge server, is of the order of 20 to 45 milliseconds (Ameur et al. (2021)). Many requests for processing resources at an edge server might be queued at overloaded times, which will result in queue delays and render impractical latency-sensitive IoT applications.

In this regard, studies in the literature have been devoted to tackling the resource management and allocation problem in edge servers for different IoT application scenarios (e.g., P. Lin et al. (2021); Pang et al. (2021)). Overall, such works aim at allocating edge resources to IoT applications based on a general fairness requirement, energy cost at IoT devices, or based on the different requirements of the distinct IoT applications. Nevertheless, the resource management and allocation problem are independently performed at individual edge nodes. Herein, we assume that a set of edge servers are deployed in the network edge and shared by a set of IoT users through different

access networks. Hence, an IoT resource request can be forwarded to the proper edge servers that have available resources to process it immediately. In this setting, the allocation and management of the edge resources must be done in a joint manner aimed at achieving an overall desired performance. The balance of IoT load at a set of connected edge servers can contribute to improving the performance of time-sensitive IoT applications since IoT requests made to an overloaded edge node are forwarded and promptly served by edge nodes with available resources.

In this chapter, we proposed a novel algorithm for balancing IoT workload among networked edge servers. The proposed algorithm runs at a software-defined networking (SDN) controller and takes advantage of the underline SDN architecture to periodically collect information, from programmable switches, of the IoT workload flows forwarded to each edge server. We propose a load score metric used to rank edge servers based on the IoT workload that each edge server is processing. Furthermore, the designed algorithm periodically re-assigns IoT flows to edge nodes, from obtained nodes load score and estimated incoming and outgoing traffic load, aimed at achieving a balanced workload at edge devices and, consequently, improved performance of IoT applications. We conducted extensive simulations using the Mininet emulator to evaluate the performance of the proposed solution with related schemes in an IoT video analytics application. Obtained results show that the proposed solution outperforms deterministic and random-based IoT video streaming flows assignments for edge servers.

The remainder of this chapter is organized as follows. Section 4.2 discusses the related work. Section 4.3 provides the mathematical model for the considered SDN-enabled edge computing system for IoT video analytics. Section 4.4 presents the proposed SDN-based solution for load balancing at edge servers aimed at reducing latency on IoT video analytics applications. Section 4.5 discusses the conducted simulation-based performance evaluation of the proposed solution and related schemes. Finally, Section 4.6 presents the final remarks and discusses future work.

3.2 Related Work

Herein, studies that proposed edge computing and SDN for supporting IoT applications are discussed.

M. Chen and Hao (2018) designed a SDN based hierarchical architecture for IoT task offloading and task distribution between the edge servers. The SDN controller maintain information about mobile devices, edge servers, as well as mobile tasks to be able to make decision on where tasks should be processed, i.e., either locally or at the edge servers aiming at reducing the power consumption of mobile devices and processing latency. In their proposed SDTO scheme, the mobile devices and edge servers will regularly send the updated information about the available battery and tasks at the mobile devices and available resources at the edge servers to be used by SDN controller to make the offloading decisions. The authors compared their developed scheme with random offloading scheme and uniform offloading scheme.

Belkout et al. (2022) referred to distributed network of fog systems, as well as the limited resources of fog nodes. To tackle with these problems, the authors developed a system named Load Balancer Smart Controller (LBSC). When a new task offloading request from an IoT device arrives at LBSC, the Selection Module inside the LBSC select a fog node with enough resources to perform the task by using Reinforcement Learning. After that, the Selection Module sends the ID of the selected fog node to the Routing Module. The Routing Module uses the Dijkstra algorithm to find an optimal route based on the link states from the IoT device to the fog node.

Misra and Saha (2019) studied the IoT task offloading problem in a software-defined access network IoT devices and fog nodes are connected through multi-hop access points. The authors designed a scheme that optimally decides when a task must be processed locally at the IoT device or offloaded to a fog node, selects the fog node to process the task, and determine the routing path to be used in the offloading process. The authors formulated the proposed task offloading decision scheme as an integer linear program (ILP), which considered the limitation of energy of IoT devices and network conditions, and a greedy solution to solve the modeled ILP task offloading decision problem.

Akbar et al. (2021) proposed a multi-objective optimization model and a machine learning-based approach for tackling the problem of selecting reliable routing paths for task delivery in an SDN-enabled IoT-fog computing scenario. The designed approach relies on fuzzy logic theory to decide the optimal routing at SDN switches, based on observed link utilization rate, link failure, and the frequency of failure, aimed at maximizing path reliability and minimizing path delay.

Q. Liu et al. (2018) proposed the FACT network orchestration algorithm to boost the performance of mobile augmented reality applications (MAR). The authors formulated the total service latency for MAR users from the frame resolution, network latency, machine learning model, and computation capabilities of the mobile and edge devices. They also designed a network orchestrator, which selects the frame resolution and decides that video frames will be processed locally or at the edge node, aimed at minimizing the service latency and maximizing the total analytics accuracy in MAR applications.

The studies mentioned above focused on the design and development of solutions to improve the IoT application performance through edge computing. The main research challenge was the decision problem of selecting the suitable place (i.e., IoT device or edge node) to process a given IoT task or data, based on a well-defined criterion (e.g., energy cost, latency, and accuracy). The novelty and contribution of this chapter is the consideration of a set of connected edge servers and the management and allocation of their resources, to IoT applications, in a joint manner aimed at a balanced processing load and, consequently, improving the performance of the IoT application.

Kaur et al. (2016) addressed the problem of the load balancing at network servers. They assumed an SDN architecture used to manage the network and compared the performance of the Round-Robin, Direct Routing based on server load, and direct routing based on server connection algorithms, which can be used to balance the load among servers. In the Round-Robin algorithm, for responding the client requests, servers change circularly. In the Direct Routing based on the server load algorithm, servers send their CPU load to the load balancer. Every time the server with a lower CPU load would be in charge of handling the client requests. In the last proposed algorithm, Direct Routing based on server Connection, servers send their active TCP connections to the load balancer. The server which has a less amount of active connection will be in charge of responding to a new request from a client.

Hamed et al. (2017) designed a server load balancing algorithm, named BWBLB, for SDN architectures. The proposed algorithm assigns incoming requests to servers based on their used bandwidth. Accordingly, the server which has used the least bandwidth would be assigned to handle an incoming processing request. The authors proposed the use of the bwm-ng software tool to obtain the bandwidth usage at each server.

The works mentioned above focused on balancing the workload at server nodes. Nevertheless, they do not tackle the unique characteristics of IoT applications and traffic, which is heterogeneous and highly dynamic. In this chapter, our proposed solution considers the stringent QoS requirements of IoT augmented reality applications when balancing the IoT workload among edge servers.

In addition, Table 3.1 represents a qualitative comparison between related articles and the proposed solution in terms of several criteria. As we can observe from table 3.1, the contribution of the proposed solution is reducing the processing latency of IoT video analytics applications with a low overhead solution.

Table 3.1: Qualitative Comparison

Article	**Resources**	**Criteria**	**Overhead**	**SDN**
Belkout et al. (2022)	Fog	End-to-end delay	High	No
Akbar et al. (2021)	Fog	Transmission latency	Low	Yes
Misra and Saha (2019)	Fog	Processing latency\Energy	Low	Yes
M. Chen and Hao (2018)	Edge	Processing latency\Energy	High	Yes
Hamed et al. (2017)	Edge	Processing latency	Low	Yes
Proposed architecture	Edge	Processing latency	Low	Yes

3.3 System Model

In this chapter, we consider a system composed of IoT devices that offload part of their task and data to be processed on a set of edge devices, as shown in Fig. 3.1. Unless otherwise specified, we consider an IoT augmented reality application where part of IoT video frames is processed on edge nodes. This would represent those application scenarios in which edge devices can process some of the video frames for detecting objects of interest, while IoT devices would locally process part of the video frames to track the movement of the object. This would reduce the service latency experienced by the application (Hanyao et al. (2021)). We assume a software-defined network architecture that will manage the traffic flow from IoT devices to edge nodes.

The considered system consists of a set $\mathcal{I}$ of IoT devices, e.g., smart cameras deployed on an industrial floor, a set of access points $\mathcal{A}$, set of software-defined network switches $\mathcal{S}$, and a set of edge servers $\mathcal{E}$. The network is modeled as a direct graph $\mathcal{G} = (\mathcal{I} \cup \mathcal{A} \cup \mathcal{S} \cup \mathcal{E}, \mathcal{L})$, where $\mathcal{L}$ denotes the set of links. A link $l_{ij} \in \mathcal{L}$ exists if the entity i is directly connected j ($i, j \in \{\mathcal{I} \cup \mathcal{A} \cup \mathcal{S} \cup \mathcal{E}\}$).

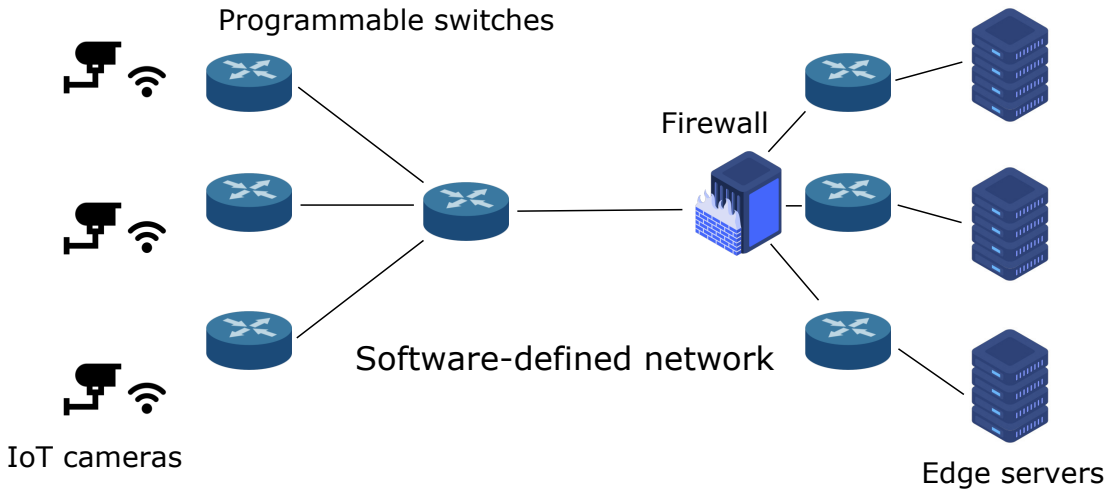


Figure 3.1: Considered network architecture

Each IoT device $i \in \mathcal{I}$ is connected to an edge server $\mathcal{E}$ through an access point $a \in \mathcal{A}$ and a set of network core switches $\{s_1, \ldots, s_n \in \mathcal{S}\}$. Furthermore, we assume that IoT device $i \in \mathcal{I}$ stream video frames, at one of the possible qualities, $\mathcal{Q} = \{q_1, \ldots, q_{|\mathcal{Q}|}\}$. Video frames are streamed by IoT devices and processed at one of the edge servers $e \in \mathcal{E}$ located in its proximity. Each quality $q \in \mathcal{Q}$ is represented by frame resolution of $r_q \times r_q$ pixels. The use of high-resolution frames leads to high accuracy by machine learning models when detecting and identifying objects of interest from received frames. However, the cost incurred at IoT devices to stream high-resolution video is high, as well as the latency to process such frames at edge servers.

We assume that a given edge node will serve different IoT devices by processing different quality video frames received from IoT devices. In a traditional IoT video analytic application, video frames are processed in a real-time manner, and the result is used to support decision-making. Herein, we consider an IoT video analytic application where objects of interest must be detected and recognized from video frames. In this application, a deep learning model, such as convolutional neural networks (CNN), can be used at edge servers to detect and recognize objects of interest from video frames received from IoT devices. In a given edge server, video frames will be processed on a first-come-first-serve basis. We assume that the computation latency for a video frame of $r_q \times r_q$ pixels can be estimated as (Q. Liu et al. (2018)):

$$f(r_q) = 7 \times 10^{-10} r_q^3 + 0.083, \tag{1}$$

Algorithm 1 Periodic Flow Re-Assignment

1: **for all** $s_i \in \mathcal{S}$ **do**
2: Obtain the number of bytes received and forwarded through each port
3: **end for**
4: Calculate the load score of each edge server $i \in \mathcal{E}$ through Eq. 3
5: **for all** IoT video stream flow F **do**
6: Select the edge node to process F based on Eq. 4
7: Update the load score of the selected edge server by adding to it the average load of F given by Eq. 5
8: **end for**=0

which is a convex function that well characterizes the observed video frames processing latency when YOLO (Redmon et al. (2016)) and SSD (W. Liu et al. (2016)) object recognition algorithms were used in a workstation with Nvidia Quadro M4000 GPU.

Besides, a frame arriving at an edge server might need to wait in a queue before being processed. Let's assume a queue L at the edge server in which frames will wait before being processed. Therefore, a frame k of quality q, i.e., $r_q \times r_q$ pixels, offloaded by the IoT device i to an edge server e, will have an overall amount of computation time, i.e., time from the moment it arrived in an edge server to the moment that the server finished its processing, when l frames are already in L, of:

$$t^i_{k,q} = f(r_q) + \sum_{\forall l \in L} f(r_{q_l}). \tag{2}$$

3.4 Proposed Solution

In this section, we discuss the lightweight proposed solution aimed at reducing the processing latency of video frames. The proposed solution aims at balancing the load at edge servers to reduce the amount of time a frame waits before being processed. The designed solution takes advantage of the underline SDN infrastructure used to deliver video frames from IoT devices to servers at the edge of the network. The general idea is to forward incoming video frames to the underloaded edge server, based on the traffic load observed in a previous time window. Algorithm 1 presents the proposed solution for balancing the IoT flows at edge servers. The proposed solution is discussed in detail in the following.

Overall, an SDN controller can periodically obtain several details (e.g., traffic flows, amount

of bytes received and forwarded, and packet drop ratio) from network switches under its domain through the OpenFlow protocol. Hence, the proposed solution relies on these periodic details for balancing the load among edge servers. The overall idea is to periodically determine the edge node that will serve a traffic flow, i.e., IoT video transmission, based on the predicted amount of data that will be transmitted in a given time window and the current traffic load at the edge nodes. To do so, time is divided in windows of the duration of $T = 30\,\text{s}$. The parameter T is called reassign time since the controller will make decisions of reassigning traffic flows based on edge servers' workloads.

Therefore, at the beginning of each time window, i.e., reassign time, programmable switches report to their SDN controller the number of bytes they have received and forwarded through each port. Such details are used by the SDN controller to estimate the traffic load at each edge node and calculate a load score for each edge server, used to rank them when determining what would be the destination, i.e., edge node for each IoT video streaming traffic flow for the next time window. The load score of an edge server $s \in \mathcal{E}$ at the time window k is estimated by the SDN controller as:

$$\varphi_i(k) = \varphi_i(k-1) + [\beta \times \boldsymbol{R}_i + (1-\beta) \times \boldsymbol{T}_i], \tag{3}$$

where $\varphi_i(k-1)$ is the score of the edge server i at the previous time window, where $\varphi_i(0) = 0$, and R_i and T_i are the total amount of bytes received at and transmitted by the edge server i, respectively. The amount of transmitted and received bytes at an edge server i is obtained by the controller from the switch $m \in \mathcal{S}$ in which the edge server is connected. The constant β is used to weigh the importance of incoming and outgoing traffic at the edge node during the process of estimating its score. Herein, β is set as 0.5 if not otherwise specified.

At the beginning of each time window k, the flow table of each programmable switch is reset. Hence, upon the reception of a packet from an IoT video streaming traffic flow f, a switch contacts the SDN controller for routing decisions regarding the incoming packets. The SDN controller selects the edge node to process the considered flow F based on Eq. 4 and installs flow rules in the switches.

$$s* = \arg\min_{s \in \mathcal{E}} \{\varphi_s(k)\}. \tag{4}$$

Subsequently, the controller updates the load score of the selected edge server by adding to it the average load μ_f of the F IoT video streaming flows admitted in the network, which is given as:

$$\mu_f = \frac{1}{F} \sum_{\forall j \in \mathcal{E}} \varphi_j(k-1) \tag{5}$$

The update on the load score of the selected server is to consider the newly assigned traffic load to the server in case switches request routing decisions for other flows during time window k.

3.5 Performance Evaluation

3.5.1 Simulation Settings

We evaluate the performance of the proposed load balancing solution using the POX SDN controller and Mininet network emulator. In the simulations, we consider the network topology shown in Fig. 3.1. We assume that IoT devices will be connected to one of the three switches located at the border of the network. The number of IoT devices connected to each border switch will vary from 1 to 7. Moreover, each IoT device will offload video frames at the rate of 10 fps. This rate represents the frames that each IoT device will offload to edge nodes while the remaining frames would be processed locally at the device. Three edge servers are considered for the video analytics of IoT video frames.

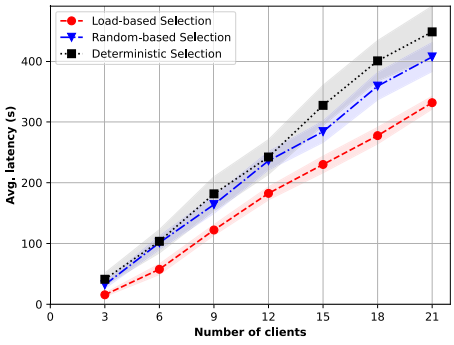
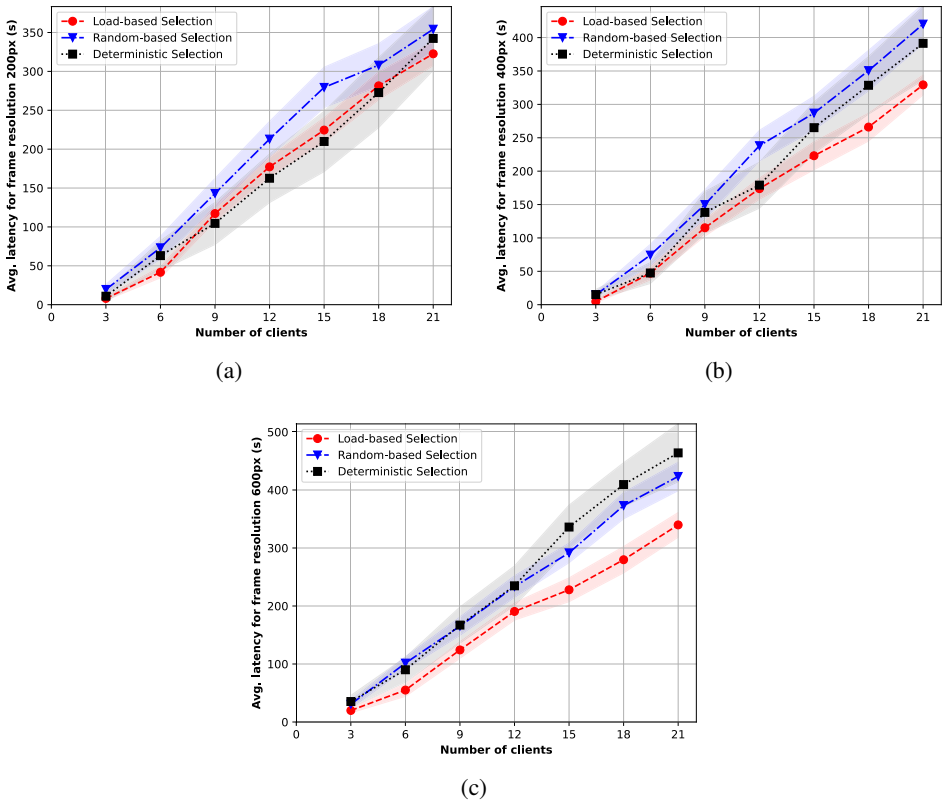


Figure 3.2: Average latency per frame

Table 3.2: Simulation parameters

Parameter	**Value**
Number of clients per border switch	[1,7]
Number of edge servers	3
Video frame resolutions (r_q)	{200px×200px, 400px×400px, 600px×600px}
Frame rate	10 fps
IoT video flow duration	100 s
Reassign time (T)	30 s
β	0.5



(a) (b)

(c)

Figure 3.3: Average latency per frame resolution: a) 200px×200px. b) 400px×400px. c) 600px×600px

For each edge server, frames are processed on a first-come-first-serve basis. The processing latency for a frame is estimated by Eq. 1 and the total amount of time that will take to complete the processing of a frame in an edge server, from the moment that the frame arrives at the server to its completion is given by Eq. 2. We compare our proposed solution, named Load-based Selection,

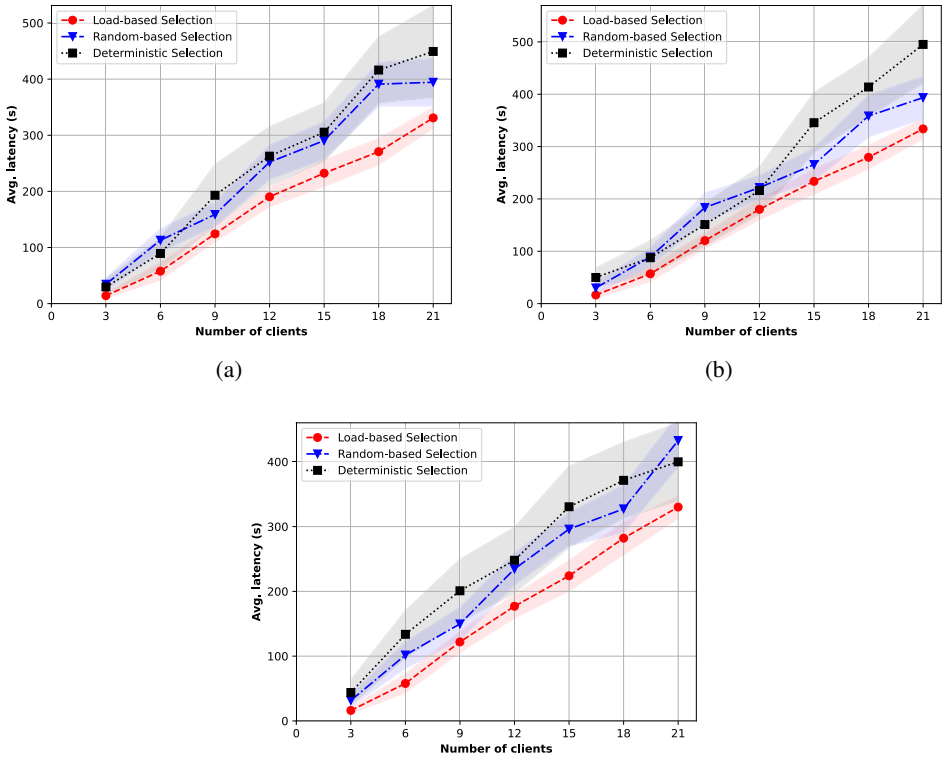


Figure 3.4: Average latency per edge server: a) Edge server 1. b) Edge server 2. c) Edge server 3

with two other schemes:

- **Random-based Selection:** In this approach, frame offloading decisions by IoT devices are completely random. That is, the edge server that will process an IoT video streaming traffic flow is randomly selected by the SDN controller at each reassign period.

- **Deterministic-based Selection:** The edge node selected to serve an IoT video streaming flow is selected randomly when the flow starts. However, the selected server will remain the same for the entire duration of the flow. That is, the server does not change during reassign times.

To study the performance of the proposed solution and related schemes, we have considered four performance metrics:

- *Average latency:* This metric represents the average amount of time to complete the analytics of a video frame from the moment it is received at one of the edge servers.

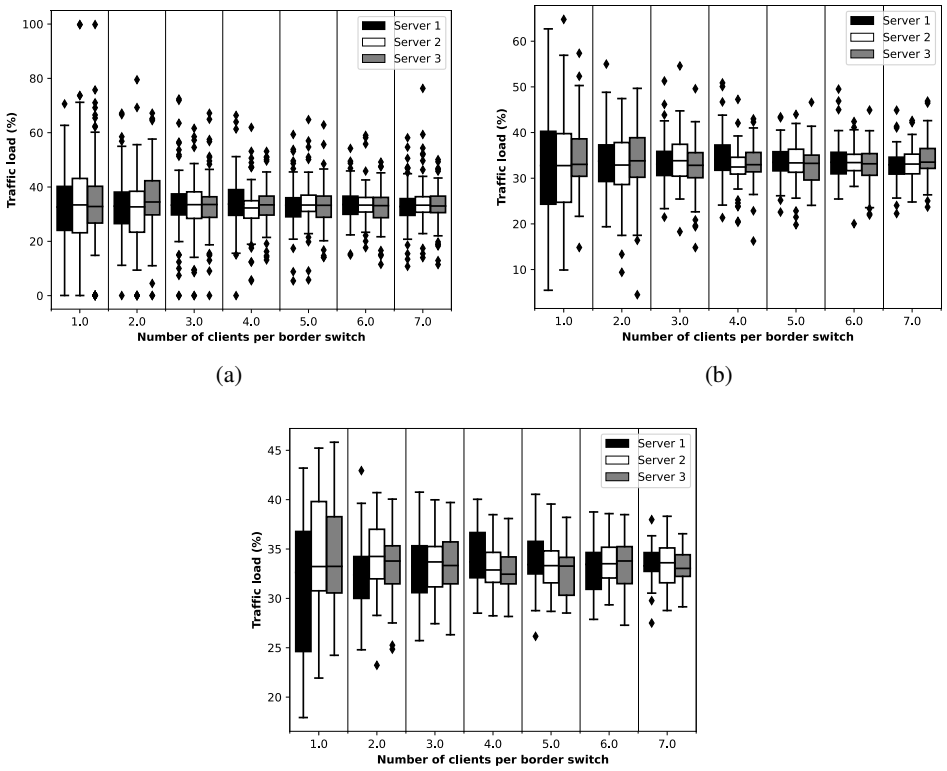


Figure 3.5: Percentage of traffic load received per server: a) Deterministic-based Selection. b) Random-based Selection. c) Load-based Selection

- *Average latency per frame resolution:* This metric represents the average amount of time to complete the analytics of a video frame of a given resolution r_q from the moment it is received at one of the edge servers.

- *Average latency per edge server:* This represents the average amount of time to complete the analytics of a frame in each server.

- *Traffic load per server:* This metric represents the perceptual of all generated traffic load, i.e., IoT video streaming frames, that was handled by each server.

The remaining simulation parameters are presented in Table 4.2. The results represent the average value of 30 replications and a confidence interval of 95 %.

3.5.2 Simulation Results

Average latency per frame

Figure 3.2 shows the average amount of time to complete the analytics of an IoT video frame as the number of IoT devices increases. First of all, it can be noticed that, as expected, the average latency for all schemes increased as the number of IoT devices increased. This increment was of 9 times when the number of IoT video streaming flows increased from 3 and 21 and no periodic flow reassignment to edge servers was performed. This happens due to the high resource demand for limited edge servers computing resources when a high number of IoT video streaming traffic flows is considered. In high load scenarios, video frames tend to have long queue waiting times before being processed.

The second trend in Fig. 3.2 is the reduced average latency when traffic flows are reassigned periodically to different edge servers. The average latency decreased even when a periodic random assignment was performed. This is because different IoT video streaming flows will consider different frame resolutions, and a periodic random selection of the edge server would reduce the average queue waiting time on them since high-resolution frames would be likely distributed among the servers. The Load-based Selection scheme designed to periodically determine the edge server for each IoT video streaming flow reduced in 21 % the average latency when compared to the deterministic edge server assignment for high IoT traffic load. This happens due to the proposed balancing load mechanism used to determine the edge node to server traffic flows at each time window.

Average latency per frame resolution

Figure 3.3 depicts the average latency experienced for frames at each considered resolution when the number of IoT video streaming flows increases. The first observed trend is the increased average latency when the frame resolution increases (see Figs. 3.3a, 3.3b, and 3.3c). Furthermore, it can be noticed that the average latency to complete the frame analytics increases, for all frame resolutions, as the number of IoT flows increases. This is already expected and corroborated by the results shown in Fig. 3.2. Those trends are due to the increased demand for edge computing

resources, which would increase the average amount of time a frame would wait in the queue before being processed. Finally, the proposed load balancing solution outperforms related schemes especially for frames using high resolution (see Fig. 3.3c). However, in some scenarios, we can observe that Deterministic-based Selection outperforms our proposed solution (see Figs. 3.3a). This is caused by the latency of calculation of edge servers scores in the SDN controller. Since in the low-resolution scenario with small number of clients, there is no significant load on each edge server, no specific load-balancing policy may be needed. However, the opposite situation is obvious when the number of clients increases (see Figs. 3.3a) or higher video frames' resolutions are streamed by IoT devices(see 3.3b, and 3.3c).

Average latency per edge server

Figure 3.4 illustrates the average latency for the analytics of an IoT video streaming frame at each server, as the number of IoT video streaming flows increases. The first trend to be noticed is that the proposed load-based solution outperforms related schemes. This is due to the periodic mechanism implemented at the SDN POX controller to reassign IoT video flows to edge servers based on the server's load. The latency decrements achieved by the proposed solution in server 3 (Fig. 3.4) were of approximately 16 % in comparison to related schemes. Moreover, it is worth noticing that the average latency at the three edge servers is similar when the designed load-based periodic edge server reassignment solution is used. This is an important aspect that contributes to the increased fairness among data analytics from different IoT video streaming flows.

Percentage of traffic load received per server

Figure 3.5 shows the percentage of the traffic load that is handled by each edge server, as the number of IoT video streaming flows increases. Overall, the average amount in the servers is similar for all solutions. However, the deterministic and random-based schemes presented several cases in which the traffic load on a server achieved more than 60 %, as can be seen in Figs. 3.5a and 3.5b, respectively. The mentioned drawback does not appear when the load-based edge node reassignment is performed periodically in the proposed solution.

3.5.3 Discussions

The observations from results provide interesting aspects of the proposed model and SDN-enabled solution. First, the decrease in the processing latency for latency sensitive applications, including IoT video analytics compared to related approaches can be a vital outcome from the developed solution.

Second, the proposed solution can integrate in dynamic environments, where computing resources as containers or virtual machines can be added or removed considering the workload which exists in the network at a certain time. This feature brings a crucial advantage in terms of managing available resources and costs as when the number of IoT devices in the network decrease. The available resources can be dedicated to other services, as well as avoiding high latency in a dense network with many IoT devices trying to join the network by increasing supporting supplies.

Besides, the results show a balanced load between the edge servers available in the network when using the proposed system. Therefore, all the available computation resources in the network can be utilized with almost the same workload to make benefits from their resources and decrease the processing latency of the offloaded task.

3.6 Conclusion

In this chapter, we proposed an SDN-enabled scheme to balance the workload at edge nodes used for real-time IoT video data analytics. The proposed scheme obtains incoming and outgoing traffic load at each edge server to estimate the traffic load and assigns a load score, which was used to re-route IoT video streaming flows periodically to achieve a balance. Simulation results showed that the proposed solution outperformed related schemes in reducing the average video frame data analytics at edge nodes through a balanced workload distribution among the servers.

Chapter 4

SDN-LB: A Novel Server Workload Balancing Algorithm for IoT Video Analytics

The widespread of Internet of things (IoT) cameras combined with powerful deep learning models for object detection and recognition will empower a new generation of IoT video analytics applications. The real-time analytics from IoT video streaming will increase industrial floor automation, improve the experience in transportation systems, increase the efficiency of autonomous and privacy-preserving surveillance systems, and unlock smart environments in smart building applications. However, IoT video analytics require intensive computing resources which are often not present in IoT devices. In this chapter, we propose a hierarchical edge/cloud-based architecture for the analytics of IoT video streaming flows. We devise a queueing model that considers the characteristics of the IoT video flows (i.e., frame rate and frame resolution), the network backbone (i.e., communication latency of the routing path from IoT devices to cloud/edge servers), and the characteristics of the servers (i.e., processing resources capabilities) to estimate the expected video frame processing latency in each server. We then propose a software-defined networking (SDN) architecture for the load balancing at the edge and cloud servers aimed at reducing the average latency for the processing of video frames. We design the SDN-LB algorithm to periodically collect

data from programmable switches, estimate the expected latency in each server, and to re-assign IoT video streaming flows to edge and cloud servers to reduce the end-to-end latency for the processing of video streaming. Extensive evaluation results show that the proposed balancing solution can effectively balance the IoT video streaming flows across the edge and cloud servers, and reduce the average latency incurred in the processing of IoT video frames.

4.1 Introduction

Recent advancements in Internet of things (IoT), wireless communication, and deep learning have empowered live video analytics for applications in different domains. In a traditional application (e.g., transportation system (R. W. Liu et al. (2022); Wan et al. (2021)), health safeguard in inhospitable environments (Singh et al. (2022)), and crowdsensing (Zhu et al. (2022))), many IoT video cameras will be deployed in areas of interest and will record video content from their coverage area. The IoT-captured videos are then processed by deep learning algorithms to detect and identify objects, animals, or people of interest and recognize actions of interest. Current architectures for IoT video analytics (e.g., Apostolo et al. (2022); Bastani et al. (2020); Kang et al. (2019); Mohamed and Zemouri (2022)) often employ the store-and-analyze approach, where the object detection and recognition are done by using deep learning models on data-at-rest video databases.

Nevertheless, a few recent approaches have been designed to provide data analytics for IoT video streaming. The overall idea consists of the deployment of many IoT cameras that will live stream video content from their coverage area. The IoT video streaming is then processed by deep learning models, promptly, for the detection and recognition of objects or gestures of interest. This real-time approach unlocks a novel realm of IoT smart applications in different domains. For instance, the mentioned system can increase automation on industrial floors where failures and imperfections on manufactured products can be promptly detected from the video streaming analytics provided by IoT cameras on the industrial floor (Coutinho and Boukerche (2022a, 2022b)). Besides, IoT video streaming analytics can be used, for instance, for monitoring and tracking transportation systems, improving the intrinsic capability of older adults in transportation systems (Yount et al. (2022)), and surveillance in airports and public spaces.

However, the design of IoT video streaming analytics is challenging. One of the daunting challenges is to ensure acceptable latency for the application. One approach is to perform the video frame processing (i.e., object detection and recognition) at the IoT camera (Fernández-Sanjurjo et al. (2021); Zeng et al. (2020)). This approach might reduce the latency since data is processed locally. However, it does not suit when high accuracy is required and the IoT devices are not powerful enough to support computation- and storage-demanding deep learning models. An alternative method consists of the offloading of video frames from IoT cameras to distant cloud servers (e.g., Poddar et al. (2020)), where powerful servers equipped with GPUs and TPUs will process received IoT video frames in a real-time manner. The drawback of cloud-aided IoT video analytics is the network overhead for the video frames offloading, which can congest routing paths and increase the experienced end-to-end latency.

Thus, a more recent approach consists of the use of multi-access edge computing, where computation, communication, and storage resources are deployed at the edge of the network, in the proximity of the IoT devices (Qian and Coutinho (2021)). This approach reduces the end-to-end communication latency and network congestion since IoT video frames are offloaded to nearby edge servers instead of distant cloud servers. However, edge servers have limited resources when compared to cloud servers. Thus, they might become overloaded, which would increase the amount of time a video frame would wait in the queue before being processed and, consequently, the latency experienced by the application.

One possible solution is to use a hierarchical cloud/edge environment, where IoT devices offload video frames to edge and cloud servers based on the experienced network condition. However, in a busy network with many IoT devices, overloaded edge and cloud servers can not be avoided, even with a decent load balancing between the edge servers. The challenge is then how to decide the number of video frames that must be offloaded to the cloud and to edge servers. This decision must be done also considering the workload at each server towards load balancing.

However, it is challenging to acquire periodic and up-to-date information from the servers to estimate their load and perform load balancing. Besides, load balancing decisions only considering the workload at the servers shall render it infeasible for IoT video analytics applications since this approach might not reduce the latency given the distinct capabilities of the edge and cloud servers.

In this chapter, we propose a novel loading balancing scheme, named SDN-LB, for hierarchical cloud/edge environment in IoT video streaming analytics. The proposed solution relies on a software-defined networking (SDN) architecture to obtain the necessary up-to-date information used to balance the load at the servers. The designed SDN-LB algorithm periodically re-assigns IoT video streaming flows across edge and cloud servers aimed at reducing the end-to-end delay experienced by the video analytics application. The contributions of this chapter are the following:

- We devise a queuing model to estimate the average latency to process a video frame in each considered edge and cloud servers. The queuing model considers the frame rate, frame resolution, number of flows, communication latency, and server processing capacity to estimate the average amount of time a video frame will spend in the server from the moment that it is received to the moment that its processing is finished.

- We propose an algorithm, named SDN-LB, to periodically estimate the workload at the edge and cloud servers, from data acquired from programmable switches, and to re-assign IoT video streaming flows across the edge and cloud servers. The proposed SDN-LB algorithm collects statistics from the programmable switches' ports through the SDN southbound API and determines the number of frames and average frame resolution that were destined for each server. Then, it uses the proposed queuing model to estimate the average latency experienced by video frames in each server. Next, it re-assigns a considered IoT video streaming flow to the server that results in the lowest processing latency.

- We conduct extensive numerical evaluations to assess the performance of the proposed solution. The proposed solution is compared to two different approaches in terms of average latency and utilization under different settings. Obtained results show that the proposed SDN-LB solution outperforms related work by decreasing the latency experienced by the frames and balancing the utilization of the network servers.

- We provide a thorough discussion of the insights obtained from the extensive numerical evaluation. We discuss how the observed main insights can drive the further design of SDN-based hierarchical edge/cloud distributed systems for real-time analytics of IoT video streaming.

The remainder of this chapter is organized as follows. Section 4.2 discusses the related work. Section 4.3 devises the proposed queuing model for estimating the average latency for the processing of IoT video streaming frames at edge and cloud servers. Section 4.4 proposes the SDN-LB algorithm designed to periodically re-assign IoT video streaming flows across edge and cloud servers aimed at reducing the average latency on IoT video analytics. We evaluate the proposed solution in Section 4.5. Finally, final remarks and future work are discussed in Section 4.6.

4.2 Related Work

In this section, we discuss in detail some recent works related to IoT architectures for video analytics, task offloading, edge/cloud computing, and SDN. In general, IoT resource-constrained devices offload frame processing tasks to be processed at edge and cloud servers. Hence, the works discussed in the following proposed solutions for tackling the offloading problem, as well as the research challenges involved in IoT communication.

M. Chen and Hao (2018) designed a three-tier framework for MEC to reduce the delay for latency-sensitive applications while considering limited energy resources for the mobile devices by using SDN. The first layer of their proposed framework is the user plane, in which users exist. The second layer named the data plane comprises the base stations (BSs) and edge/cloud servers. In the third layer, the controller of SDN exists, i.e., the control plane. The SDN controller maintains information regarding mobile devices, BSs, and tasks, which they would be used to decide on task offloading policy. They proposed an optimization technique to decide where IoT task needs to be processed, and in case of offloading to the edge clouds, how many resources the offloaded task needs.

Lyu et al. (2017) investigated the challenges incurred when a dense deployment of IoT nodes offloads their tasks to resource-constrained edge devices. The authors proposed the heuristic offloading decision algorithm (HODA) to determine whether the task needs to be offloaded or not, based on the task processing time and energy consumption when the task is offloaded compared with local execution in the node itself. The HODA algorithm consists of two steps. First, the

mobile user optimizes transmission power and decides whether the task is processed locally or being offloaded. Second, the users would be ordered based on who mostly takes advantage of task offloading. The users with high benefits are permitted to offload their tasks.

H. Guo et al. (2020) proposed a supervised learning scheme to solve the mentioned problems, named the decision tree-based offloading scheme (DTOS), which aimed to solve the optimization problem of reducing mobile devices' energy consumption and processing latency. In their developed structure, the decision tree, the leaf, and internal nodes represent classes and features respectively. Starting from the root node, DTOS tries to assign samples to the leaf nodes, i.e., classes. For the decision tree algorithm, they used C4.5 and CART and compared their effectiveness with each other.

Qu et al. (2021) referred to some problems of the Convolutional Neural Network (CNN) for task offloading decision, such as, CNN needs to reach resource devices for intensity training and if MEC environment is affected, which is possible, a lot of training data is needed to train the network again, which impose a delay to the task offloading. To solve the aforementioned and other problems, they designed a framework in which edge devices decide about task offloading. The authors converted the task offloading problem into a multi-objective problem, to minimize both delay and energy consumption. Using the combination of Deep Neural Networks (DNNs) and deep Q-learning algorithms, they proposed a framework, named Deep Meta Reinforcement learning based Offloading (DMRO), to manage the offloading decisions.

Yu et al. (2020) addressed the weakness of mobile devices to make an appropriate decision for task offloading due to the uncertain condition of the MEC environment. To this end, they proposed a deep imitation learning (DIL) model to make a decent offloading decision in the mobile devices, considering the situation of the MEC network and the consequence of processing the task in the mobile device. In their provided model, each application is divided into separate sub-tasks. Based on the authors' proposed model, The mobile device can decide for each of the sub-tasks to whether process its sub-task locally or offload it to the edge or cloud server.

He et al. (2018) proposed that since networking, caching, and computing are metrics that impact vehicular networks, considering them in separate studies may not result in an optimal outcome. Therefore, by using SDN and information-centric networking (ICN) they introduced a framework, which can enable networking, caching, and computing in a joint manner. The authors used deep

reinforcement learning to solve the optimization problem, Because the aforementioned metrics are considered jointly, they would increase the complexity of the framework.

The above-mentioned works focused on developing an optimum task offloading policy. However, when the MEC environment changes, which often happens, their frameworks need to be adapted to the new environment. In addition, in the aforementioned works, online training needs some time. Both the mentioned problems impose a delay on the IoT applications. However, in our proposed solution, there is no need to adaption in case of environmental change. Our designed algorithm, by benefits from SDN controller characteristics, can make a decent offloading decision, without any concern for the MEC environment's changes.

J. Liu et al. (2016) developed an architecture to minimize the processing delay and energy consumption with a comparison of local or MEC server processing. The authors used a one-dimensional search algorithm to find out an optimal offloading policy for each task, whether to transmit it to a MEC server or execute it on the local CPU. The mobile devices take an offloading decision based on the queuing state of the task buffer, local CPU, and communication status.

X. Guo et al. (2016) introduced a task offloading policy to optimize two metrics in the edge cloud systems, i.e., the latency that a user experiences and power consumption in the edge servers in an online fashion. In their introduced system, the mobile devices send their tasks to base stations (BSs), then the BS will send the task to its associated edge server. To propose an optimal task offloading policy, the authors used the Markov decision problem (MDP), and also to overcome the computation complexity of the MDP, they used the index policy method.

K. Zhang et al. (2016) developed a mechanism for task offloading and radio resource allocation in a MEC environment, to minimize energy consumption while considering the latency constraints of IoT tasks in a 5G network. To solve the optimization problem, the authors introduced a scheme, named Energy-Efficient Computation Offloading (EECO). The EECO scheme has three stages, and in these stages, EECO prioritised the mobile devices and based on mobile devices' priority, edge clouds' resources are allocated to them.

The main research challenge of both the aforementioned studies was to propose an offloading policy to reduce energy consumption as well as latency. Nonetheless, since mobile devices are responsible for task offloading policy, they need a periodic update from edge servers and the MEC

environment. The mobile devices with limited energy and computation resources would be exposed to the extra load. In our proposed solution, however, the nodes in the network do not resolve on offloading the IoT tasks. The SDN controller makes decisions, which because of its global view can gather useful information regarding the nodes in the network.

Roy et al. (2017) proposed a three-layer architecture for IoT task offloading based on the type of the mobile application. They addressed the problem of the overloaded cloudlets, which is a consequence of selecting one specific cloudlet by the mobile devices instead of choosing other under-loaded cloudlets. In their proposed approach, based on the application type, a specific cloudlet will be selected, aimed at reducing computation latency and energy consumption. If the mentioned cloudlet can not process the task, it will offload the task to the cloud. Then, the result will be sent back to the mobile device through the cloudlet. However, in a dense IoT network, cloudlets may become a bottleneck since the tasks and results should be sent through the cloudlet. In addition, because each cloudlet is specific to an application, we may face unbalancing workload in cloudlets, depending on mobile devices' applications.

Amjad et al. (2017) designed a hierarchy framework, named DTMO, to distribute the processing tasks between cloudlets and cloud servers to reduce the computation time and energy consumption. In their proposed framework, the IoT devices will offload their tasks to their corresponding local processing platform (LPP). That is, each LLP is in charge of a group of specific IoT devices. By considering the computational resources needed for the offloaded task and the available resource and energy at the LLP, the LLP may decide to offload the task to a cloud server. If the cloud server had the required resources, the offloaded task will be processed, otherwise, it will be offloaded to another LLP that has available resources for completing the task. Nonetheless, their proposed solution would impose an extra delay in case of the cloud server doesn't have the available resources, since again it would be offloaded to another LLP server. Moreover, if there exists an LLP with the required resource, it may be a better option to offload the task to the mentioned LLP instead of the cloud servers

S. Li et al. (2018) developed an architecture, named ECloT, to enhance IoT task offloading. Considering the benefits of offloading the tasks to the fog nodes, their architecture will offload IoT tasks to the fog nodes in the first step. In the case of insufficient available resources in the fog nodes,

the task will be offloaded to the cloud servers to be processed. The authors assumed SDN is used in the underlying network to reduce the ECIoT complexity and find an optimum route between fog nodes and cloud servers. Also, the SDN controller with its global view helps their architecture to gather the required information. Although their work seems promising, they didn't consider the edge and cloud servers jointly. When there exist many task offloading requests, maybe offloading the tasks to a cloud server in the first step is a better solution to decrease the delay.

Wei et al. (2017) proposed an architecture, named MVR, for IoT task offloading to reduce energy consumption and computation latency. In summary, their solution splits the IoT tasks into several sub-tasks, named Tasklet. Then, these tasklets would be analyzed by a segment of the mobile device that which of them need to be executed at the mobile device, and the others should be processed at the edge cloud. At the edge cloud the tasklets would be separated between different virtual resources (VRs) and one specific VR, VRs Controller (VRC), will gather the result and send it back to the mobile device. The limitation of their work is that they didn't mention how a mobile device will select an appropriate edge server. This may lead to an overloaded edge server.

Ali and Alagan (2019) introduced a mechanism to not only consider the queue delay in the edge servers but also reduce the power consumption by the edge servers. They modeled edge servers by the M/M/K queuing system, and therefore, calculated the average queuing delay that an edge server may experience. Thereafter, by using SDN, their solution turn off as much edge cloud as possible so that the queue delay does not exceed a threshold. In addition, they used a discrete-time Markov chain algorithm to manage the load balancing between the active edge servers, regarding the task offloading from the sleeping to active edge servers. However, their proposed mechanism did not consider network latency. All the edge servers near an IoT device may turn off, and therefore, network latency would be experienced. Moreover, the novelty of our algorithm is that we considered the cloud server as a powerful destination in a different possible network situation.

Q. Liu et al. (2018) proposed a system for mobile augmented reality (MAR) in the edge computing networks. The authors addressed the trade-off between network latency, computation latency, as well as analytics accuracy. Based on their MAR testbed, they modeled network latency, computation latency, and analytics accuracy. In addition, they developed an algorithm, named fast and accurate object analytics (FACT), which solves the optimization problem for trading-off between

network latency, computation latency, and analytics accuracy.

In contrast, this chapter proposes a hierarchical edge and cloud environment for IoT video analytics. It proposes an SDN architecture to periodically obtain traffic data forwarded to each considered edge and cloud server. It devises a queue-based mathematical framework for estimating the average processing latency required to process video frames at the edge and cloud servers. It also design a novel algorithm for IoT video streaming flows re-assignment aimed at reducing the latency experienced by IoT video frames.

Wu et al. (2021) introduced an architecture, called EEDTO, to offload IoT tasks to edge or cloud servers or compute them locally based on the type of the IoT applications, i.e., delay sensitive or computation intensive applications and energy constraints of IoT device using Lyapunov optimization. The authors, also addressed the security problems of task offloading and proposed a blockchain network to prevent privacy leakage, where IoT devices are miner and IoT tasks are mining tasks. Performance of EEDTO is compared with IoT-only scheme, edge-only scheme, cloud-only scheme, and lagrangian relaxation-based aggregated cost scheme.

Meng et al. (2018) addressed timing attack vulnerability in IoT task offloading in which IoT devices tries to offload their tasks to a cloud server, and attacker tries to get access to RSA private key. The authors introduced a system using hybrid Continuous-time Markov chain and queuing model to consider the trade off between security improvement by adding random delay as well as regenerating the private key and performance. Since key regeneration impose extra latency to the system as the IoT devices are not able to offload their task, their system calculates an optimal re-keying interval to optimize between security and performance.

J. Li et al. (2020b) developed an offloading algorithm in MEC environment. The authors modeled communication between edge server and mobile device, computation latency and energy consumption whether the task is processed locally or at edge server, and task queue at edge server. Their proposed algorithm uses Game Theory to make decision to compute task locally at mobile device or offload it to the edge server based on energy consumption and latency constraints of task.

J. Liu et al. (2022) proposed a system to maximize reliability while minimizing bandwidth consumption in IoT task offloading for delay sensitive applications in which tasks are modeled as

a directed acyclic graph. In their system, edge cloud servers are modeled to have certain number of VMs with failure and recovery rate. Then, the authors modeled bandwidth consumption, transmission delay, and reliability. Two different optimization algorithms, i.e., reliability-enhanced task offloading approach and differential evolution based task offloading approach are introduced to reach an optimal offloading solution based on the bandwidth and reliability.

All the above mentioned articles proposed algorithms for IoT task offloading and load-balancing. However, the contribution of our designed schema is introducing a lightweight algorithm to distribute the offloaded IoT tasks between edge and cloud servers in a hierarchical edge\cloud architecture as can be seen in table 4.1.

Table 4.1: Qualitative Comparison

Article	**Resources**	**Criteria**	**Overhead**	**SDN**
J. Liu et al. (2022)	Edge	Transmission latency	High	No
J. Li et al. (2020b)	Edge	End-to-end delay\Energy	High	No
Qu et al. (2021)	Edge	Processing latency\Energy	High	No
H. Guo et al. (2020)	Edge	Processing latency\Energy	High	No
Ali and Alagan (2019)	Edge	Processing latency\Energy	Low	Yes
Amjad et al. (2017)	Cloudlet\Cloud	Processing latency\Energy	Low	No
Lyu et al. (2017)	Edge	Processing latency\Energy	High	No
Roy et al. (2017)	Cloudlet\Cloud	Processing latency	High	No
Proposed architecture	Edge\Cloud	End-to-end delay	Low	Yes

4.3 System Model

We consider the hierarchical deployment of edge and cloud computing infrastructure used to provide computation resources for real-time analytics of IoT video streaming. In the considered system, an SDN architecture manages the network and routing of IoT video streaming flow to the cloud and edge servers. We define the set $\mathcal{I} = \{i_1, \ldots, i_N\}$ of IoT nodes that will offload video frames to be processed in the edge or cloud servers. Each IoT device $i \in \mathcal{I}$ will live stream video at resolution $q_i \in \mathcal{Q}$, where $\mathcal{Q} = \{q_1, \ldots, q_k\}$ is the set of possible video resolutions, and at a frame rate of r_i frames per seconds. The size of a video frame, in bits, of $q_i \in \mathcal{Q}$ resolution is estimated as:

$$b_{q_i} = q_i^2 \times C, \tag{6}$$

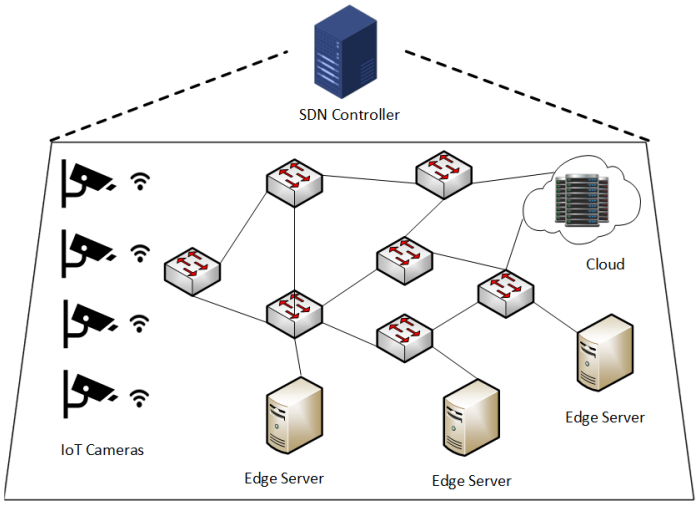


Figure 4.1: Proposed System Architecture

where C is the color depth of the image, which represents the number of bits used to encode a pixel in the frame. Unless otherwise specified, we assume the frames are in $C = 24$ bits color depth.

Let's define $\mathcal{E} = \{e_1, \ldots, e_m\}$ as the set of edge servers and $\mathcal{C} = \{c_1, \ldots, c_n\}$ as the set of cloud servers. From the edge and cloud servers, we define $\mathcal{S} = \mathcal{E} \cup \mathcal{C}$ as the set of servers in the system. We assume that edge servers are deployed in the proximity of the IoT devices, while cloud servers s_c are deployed at a distant location. We assume that the propagation delay from the IoT nodes to the edge servers is given as d_p^e and to the cloud server is d_p^c, where $d_p^e < d_p^c$. The propagation delay from an IoT node to a server is estimated from the propagation delay of individual links connecting the IoT device to the considered server.

Herein, an SDN architecture is used to manage the video streaming flows from the IoT devices to the server nodes. The SDN controller will periodically re-assign IoT flows to the edge and cloud servers aimed at balancing the workload at the servers and reducing the overall latency experienced by the IoT video analytics application. Therefore, we proposed a novel queue theory-based model used by the SDN controller to estimate the average processing latency of video frames at the different edge and cloud servers and a novel algorithm for the periodic re-assignment of IoT video analytics flows among the servers aimed at balancing the workload on them and reduce the overall latency. The proposed algorithm at the SDN controller periodically re-assigns IoT video streaming flows among the servers, based on their observed workload at the moment. We define T as the

period used at the controller to make a re-assignment decision for each specific flow, i.e., to select the server to process the flow. That is, for a given IoT video streaming flow that started at t the SDN controller will determine the most suitable server $s \in \mathcal{S}$ to process the frames at t, $t+T$, $t+2T$, $t+3T$, and so on.

Based on Yuchong et al. (2019) and Ali and Alagan (2019), each server $s \in \mathcal{S}$ is modeled as a $M/M/1$ queue. At a given time instant t when the SDN controller is making a re-assignment decision for a given flow, the arrival rate and depart (service) rate of a server $s \in \mathcal{S}$ is represented by $\lambda_s(t)$ and μ_s, respectively. We define $x_{is}(t)$ as an indicator variable where $x_{si}(t) = 1$ means that the video streaming flow of the IoT device $i \in \mathcal{I}$ is being served by the server $s \in \mathcal{S}$ at the time t, and $x_{si}(t) = 0$ means otherwise. Therefore, the data arrival rate in frames per second at the server s at time t is estimated as:

$$\lambda_s(t) = \sum_{\forall i \in \mathcal{I}} r_i \times x_{is}(t), \tag{7}$$

where r_i is the device i's video's frame per second. There has been a lot of research in the context of network slicing and application-aware SDN controllers. Using these methods, we consider that the SDN controller is aware of the resolution $q_i \in \mathcal{Q}$ which is used by the IoT device $i \in \mathcal{I}$. In addition, by benefits from the global view of the SDN controller and Openflow protocol, we can gain access to the statistics information, related to the IoT devices, such as the number of bytes that have been sent by a specific flow, or in other words by a specific IoT device $i \in \mathcal{I}$. Considering the aforementioned information, the frame rate of videos streamed by IoT devices can be calculated as:

$$r_i = \frac{f_i}{b_{q_i}}, \tag{8}$$

where f_i is the number of bits that have been transmitted by a flow related to the IoT device $i \in \mathcal{I}$.

Herein, we rely on the experimental results obtained in Q. Liu et al. (2018) and Ameur et al. (2021) to determine the service rate capability of edge and cloud servers in the modeled IoT AR system. The mentioned studies performed measurements of object detection on real devices, under different devices capabilities, frame resolutions, and deep neural network models. Accordingly, the latency (in ms) for processing a frame of resolution q_i can be approximated by the following cubic

regression:

$$\psi(q_i^2) = a + b \times q_i^3, \tag{9}$$

where a and b are coefficients set based on the different types of processing units of the servers. Herein, unless otherwise specified, we set $a = 3.23$ and $b = 9.56 \times 10^{-10}$ for cloud servers and $a = 20.98$ and $b = 3.37 \times 10^{-9}$ for edge servers, respectively.

In Eq. 9, the frame processing latency is given as a function of the frame resolution. However, edge and cloud servers will receive video frames from different IoT video streaming flows, which can be encoded under different resolutions. It is necessary to find the average resolution of frames processed in a server $s \in \mathcal{S}$, to estimate the average latency for the frames processed at that server. To do so, let's define the indicator variable $y_{iq} = 1$ to indicate that the IoT device $i \in \mathcal{I}$ is streaming video at the resolution $q \in \mathcal{Q}$ and $y_{iq} = 0$, otherwise. Hence, the average resolutions of frames processed at a given server $s \in \mathcal{S}$, within a given interval T, can be estimated as:

$$\overline{q}_s(T) = \frac{\sum\limits_{\forall q \in \mathcal{Q}} \sum\limits_{\forall i \in \mathcal{I}} x_{is} \times y_{iq} \times q}{\sum\limits_{\forall i \in \mathcal{I}} x_{is}}. \tag{10}$$

The average latency experienced by frames processed during the interval T at the server s can be estimated from Eq. 9 and Eq. 10, as:

$$\psi_s(\overline{q_s}^2) = a + b \times \overline{q_s}^3. \tag{11}$$

Hence, the service rate of the server s during the time interval T is $\mu_s(T) = 1/\psi_s(\overline{q_s}^2)$. Therefore, the utilization of the server s during an interval T is estimated as:

$$\rho_s(T) = \frac{\lambda_s(T)}{\mu_s(T)}. \tag{12}$$

Within an interval of T, from the M/M/1 queue model, the average time that a frame spends in a server can be calculated as:

$$E(T) = \frac{1/\mu_s(T)}{1 - \rho_s(T)}. \tag{13}$$

Finally, in an interval T, the total latency experienced by a frame from the moment it is transmitted by the IoT device to the moment its processing is finished at the server s is estimated as:

$$D_s(T) = d_p^s + E(T), \tag{14}$$

were d_p^s is the communication latency involved in the process of data delivery from the IoT device to the server s. This latency is given by the sum of the latency of the individual links between the routers from the IoT device to the destination server.

From Eq. 14, the average latency from when the streamed video left the IoT device to when the processing is finished in either edge or cloud server can be calculated as:

$$L(T) = \frac{1}{S} \sum_{s=1}^{\mathcal{S}} D_s(T). \tag{15}$$

4.4 The Proposed SDN-LB Algorithm

One of the daunting challenges in IoT video analytics is the high latency for video frame processing. In order to reduce latency, we propose a hierarchical cloud/edge environment where video frames from IoT devices' live streams are processed at edge and cloud servers. The general idea is to process video frames at edge servers deployed in proximity to reduce the latency incurred for offloading video frames to distant cloud servers. However, edge servers might become congested as their demand increases. Thus, part of the video frames can be offloaded to cloud servers aimed at demand at edge servers during peak time. In this regard, a dynamic solution is required to periodically monitor the workload at the edge servers and determine the destination, i.e., the server that will process each of the video streaming flows.

Therefore, we use a software-defined networking architecture, since it can decouple the data plane from the control plane. In this work, the considered SDN architecture, as shown in Fig. 4.2, will provide the management of the network elements, i.e., forwarding devices, for the routing of each flow from the sender IoT devices to the server $s \in \mathcal{S}$ selected to process it during the upcoming time interval T. Herein, the use of an SDN architecture is also advantageous given the provisioning of the means for the autonomous balancing of the workload, through our proposed load balancing

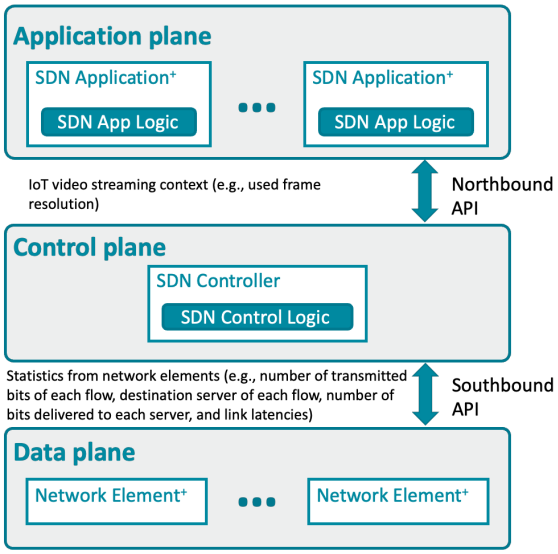


Figure 4.2: Considered SDN Architecture

algorithm, when an edge server or cloud server is added to the system.

In the proposed solution, IoT devices' video streaming will be represented as SDN applications hosted in the application plane. Each SDN application will request the controller to determine what server (edge or cloud) should be used to process the frames of the IoT video streaming and what is the routing path that the frames should take to reach the selected destination. To do so, an SDN application will maintain at least a minimal context of the represented IoT video streaming, such as the used frame resolution q_i^2 and the used color depth C in the image. The SDN controller will implement the proposed load balancing algorithm, which relies on the queuing-based model proposed in Section 4.3, to periodically re-assign IoT flows to edge and cloud servers. This re-assignment is done to balance the workload at the servers and reduce the latency experienced by the frames. The SDN controller uses the southbound API for obtaining statistics from managed network entities. For instance, it will obtain the number of bits transmitted in each flow during the time interval T from the ports of the network elements (switches) where the IoT devices are connected, and the number of bits forwarded to each server $s \in \mathcal{S}$ within the time interval T, from the network elements in which the servers are connected. From collected statistics, the SDN controller will determine the server s that will process the video frames from a flow p transmitted by the IoT device i, aimed at balancing the workload at the servers and reducing the latency experienced

by the IoT devices.

The SDN controller periodically request an OpenFlow switch by a port status request message to receive information about all or a specific port. The OpenFlow switch answers the SDN controller with an event type packet and specify information, such as number of received and transmitted packets, dropped packets, as well as transmitted and received bytes about requested ports. The size of each post status request is 80 byte, and the size of replied packet is 112 byte.

Algorithm 2 presents the load balancing procedure performed at the SDN controller. The load balancing is performed by the periodic re-assignment, i.e., the selection of the destination, of the IoT flows among the edge and cloud servers. The SDN controller, when re-assigning a flow p, will first obtain the context information for the flow from the corresponding SDN application hosted in the application plane (Line 4). The next step will then be to obtain the statistics from the network elements within the SDN controller domain (Line 5). If the flow p is new, i.e., the IoT device just connected to the system and started streaming live video, the flow will initially be assigned to the cloud server (Line 7) and a re-assignment timer for the flow p is set (Line 26). This flow p will then be considered again for the re-assignment procedure when T_p expires.

If p is not a new flow, i.e., it is already being served by one of the edge or cloud servers in the network, it might need to be re-assigned to a new server for load balancing. Thus, the first step is to estimate the incoming traffic load at the servers, based on Eq. 7. This procedure is implemented in Lines 10 to 15. Besides, the load produced by the flow p is also contemplated in the considered server s (Line 16) to estimate its load if the flow p is assigned to it. Hence, the average resolution of the video frames being processed in the server s is estimated (Line 17). This information is used to estimate the average latency to process received video frames at the server (Line 18). The next step is then the calculation of the server utilization (Line 19), which is used to estimate the average amount of time a received frame will spend in the server (Line 20), i.e., waiting time and processing time. The total latency experienced by received frames, which includes the communication latency from the IoT devices to the server, is estimated for the considered server s (Line 21). The server s^* that will present the lowest latency will be selected and assigned to process the video frames generated by the flow p (Lines 23 and 24). Finally, a re-assignment timer T_p is then set (Line 26), where the controller considers the flow again in the re-assignment procedure.

Algorithm 2 SDN-LB Flow Re-Assignment

procedure FLOWRE-ASSIGNMENT(p, i)
{p: flow to be considered.}
{i: the IoT device in which the flow is being considered.}
Obtain flow context information
Obtain updated information regarding IoT flows from the programmable devices within the SDN controller domain
if p is a new IoT flow **then**
    Assign flow p of IoT device i to the cloud server
else
    for $s \in \mathcal{S}$ **do**
        $\lambda_s(t) \leftarrow 0$
        for $j \in \mathcal{I} \setminus \{i\}$ **do**
            Find b_{q_j} according to Eq. 6
            Find r_j according to Eq. 8
            $\lambda_s(t) \leftarrow \lambda_s(t) + r_j \times x_{js}$
        end for
        $\lambda_s(t) \leftarrow \lambda_s(t) + r_i$
        Calculate $\overline{q}_s(T)$ according to Eq. 10
        Calculate $\psi_s(\overline{q_s}^2)$ according to Eq. 11
        Calculate $\rho_s(T)$ according to Eq. 12
        Calculate $E(T)$ according to Eq. 13
        Calculate $D_s(T)$ according to Eq. 14
    end for
    $s^* = \arg\min_{s \in \mathcal{S}} \{D_s(T)\}$
    Assign flow p from IoT device i to server s^*
end if
Set re-assignment time T_p for the flow p
end procedure =0

4.5 Performance Evaluation

In this section, we evaluate the performance of the proposed load balancing algorithm discussed in Section 4.4 to reduce the latency which will be experienced in the real-time video streaming applications. We conduct an extensive numerical evaluation to assess the performance of the proposed solution against two other approaches discussed below:

- **Cloud-only Approach (CoA):** In this approach, the cloud server will be selected to process the video frames. That is, the controller will forward all the incoming video streaming traffic flows to the cloud server.

Table 4.2: Numerical Evaluation Parameters

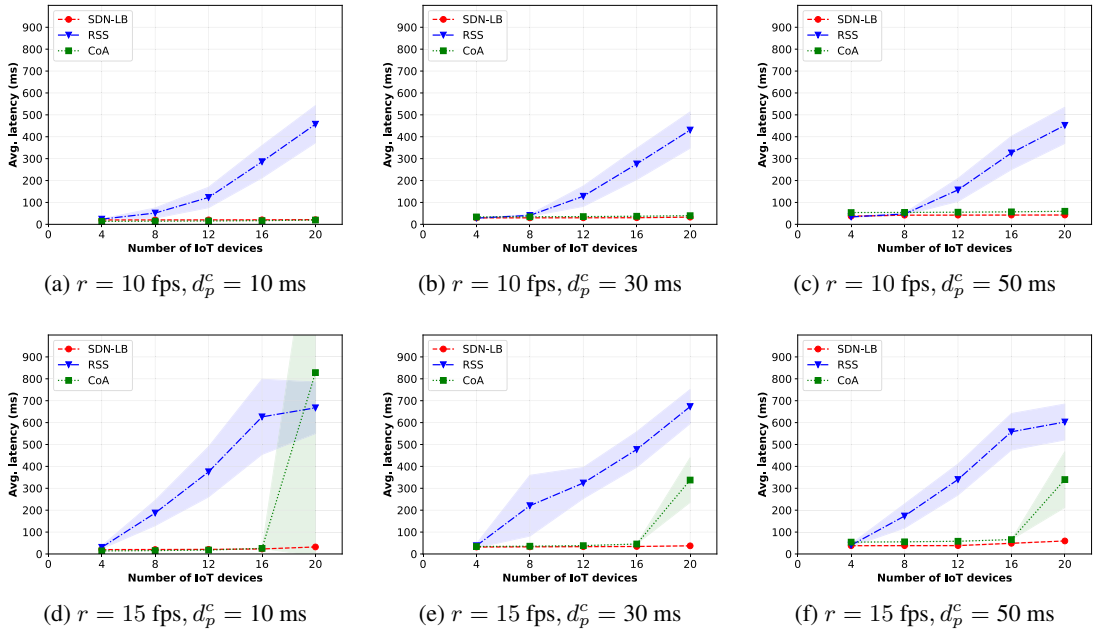
Parameter	Value
# of Edge Servers	3
# of Cloud Servers	1
Delay of WAN (d_p^c)	$\{10, 30, 50\}$ ms
Delay of WLAN (d_p^e)	1 ms
Frame rates r	$\{10, 15\}$ fps
Flow re-assignment time T	10 s
Flow duration	300 s

- **Random Server Selection (RSS):** In this approach, a server $s \in S$ will be randomly chosen to process the offloading video analytics in a considered flow p. In other words, the controller will direct IoT streaming frames to one of the servers to process the streaming video. The server, which is in charge of the video frames of an IoT device, will not change in each reassign time.

4.5.1 Scenario's setup

We used the Python programming language to implement the system model proposed in Section 4.3 and the load balancing schemes, i.e., the SDN-based load balancing algorithm proposed in Section 4.4, the CoA, and RSS approaches. We consider a scenario where one cloud server and three edge servers are available for IoT video streaming analytics. The computation capability of the servers is set in terms of the latency they take to process a video frame. It is set based on Eq. 9 devised from experiments conducted in Q. Liu et al. (2018) and Ameur et al. (2021), where $a = 3.23$ and $b = 9.56 \times 10^{-10}$ for the cloud server, and $a = 20.98$ and $b = 3.37 \times 10^{-9}$ for edge servers.

We have considered a varied number of IoT devices from 4 to 20. We consider a flow re-assignment time T of 10 s. The number of re-assignments that will be done for each flow is $n = V_d/T$, where V_d is the duration of the video stream. We assume all considered IoT devices are present at the beginning of the simulation. In addition, each IoT device streams $V_d = 300$ s duration video. Besides, each IoT device will randomly chose a frame resolution from the set $q = \{100\text{px} \times 100\text{px},\ 200\text{px} \times 200\text{px},\ 300\text{px} \times 300\text{px},\ 500\text{px} \times 500\text{px},\ 800\text{px} \times 800\text{px}\}$. In our experiments, two



(a) $r = 10$ fps, $d_p^c = 10$ ms (b) $r = 10$ fps, $d_p^c = 30$ ms (c) $r = 10$ fps, $d_p^c = 50$ ms

(d) $r = 15$ fps, $d_p^c = 10$ ms (e) $r = 15$ fps, $d_p^c = 30$ ms (f) $r = 15$ fps, $d_p^c = 50$ ms

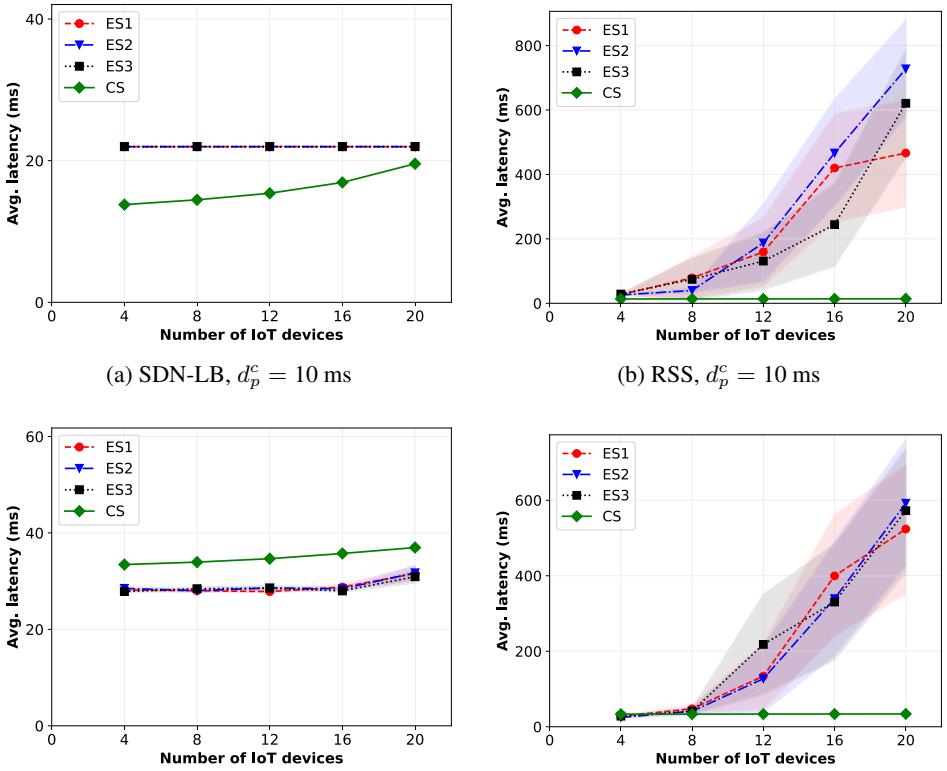
Figure 4.3: Average system's latency versus WAN delay

different frame rate $r = \{10, 15\}$ fps have been used to evaluate our model in different scenarios in comparison with other solutions. The considered frame rate would only correspond to the frames that would be offloaded to edge and cloud servers, where the remaining rate would be processed locally in the IoT device for better performance for the application. A list of used parameters can be observed in Table 4.2. The results represent the average value of 30 replications and a confidence interval of 95 %. The following metrics are used to evaluate the performance of the proposed load balancing solution and related approaches:

- **Average system latency (ms):** represents the average amount of time required to process a video frame in the system. It is calculated from Eq. 15, as:

$$L = \sum_{i=1}^{n} L(i). \tag{16}$$

- **Average latency per server (ms):** is the average latency that video frames will experience



(a) SDN-LB, $d_p^c = 10$ ms

(b) RSS, $d_p^c = 10$ ms

(c) SDN-LB, $d_p^c = 30$ ms

(d) RSS, $d_p^c = 30$ ms

Figure 4.4: Average latency per approach when frame rate is $r = 10$ fps

on each server. This metric is calculated from Eq. 14, as:

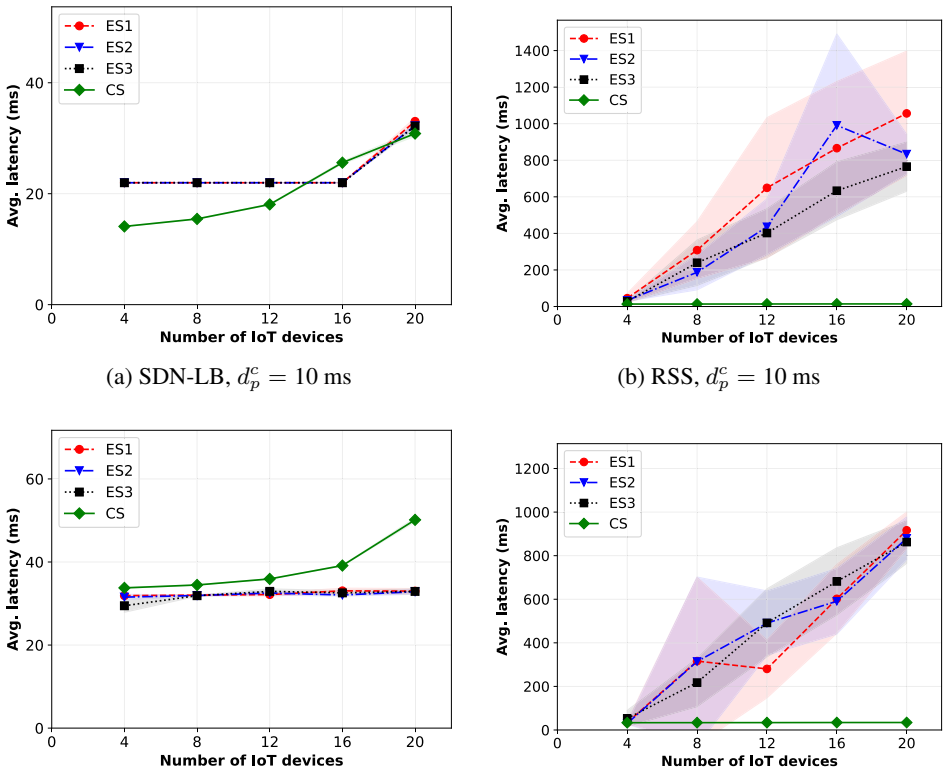
$$D_s = \sum_{i=1}^{n} D_s(i). \tag{17}$$

- **Server utilization (%):** depicts the percentage of the server capacity that is being demanded to processing of the received video frames. The utilization for each server is calculated from Eq. 12, as:

$$\rho_s = \sum_{i=1}^{n} \rho_s(i). \tag{18}$$

4.5.2 Results

Fig. 4.3 shows the average system's latency when different frame rates and latency to the server cloud are considered. The results show that the proposed SDN-LB and the cloud-only approach have similar performance when the communication latency is low (see Fig. 4.3a). This happens



(a) SDN-LB, $d_p^c = 10$ ms

(b) RSS, $d_p^c = 10$ ms

(c) SDN-LB, $d_p^c = 30$ ms

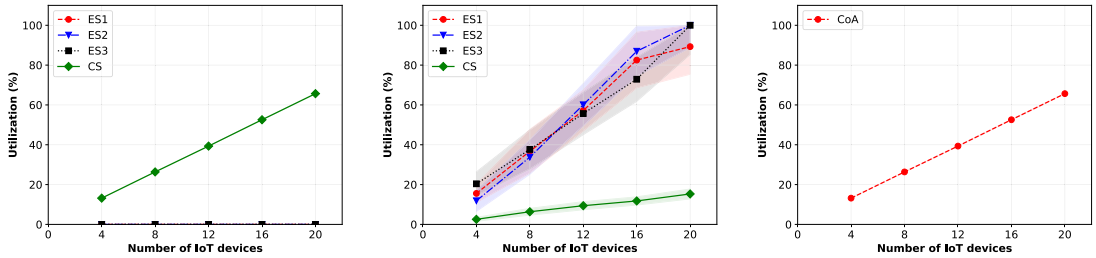
(d) RSS, $d_p^c = 30$ ms

Figure 4.5: Average latency per approach when frame rate is $r = 15$ fps

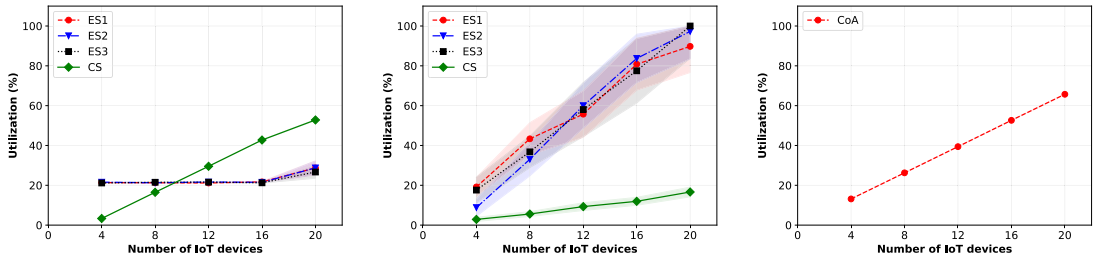
because the processing capability of the cloud server compensates for the considered communication latency. Nevertheless, as the communication latency to the cloud server increases, the proposed SDN-LB solution outperforms the cloud-only approach (see Fig. 4.3c). As expected, the RSS approach presented the worst performance the latency or the servers' capacity are not considered during the flow re-assignments.

As the workload increases, i.e., the video frame rates, the performance of the RSS approach get worse (see Figs. 4.3d, e, and f). The latency in the cloud-only approach increases as well when the number of IoT devices increases. This is because the cloud server becomes overloaded and many video frames must wait longer in the queue. In contrast, the proposed SDN-LB solution maintains the overall latency at a low level even when the workload increases. This is due to the load balancing mechanism that periodically re-assigns traffic flows among the edge and cloud servers based on the experienced latency in each server.

Fig. 4.4 depicts the average latency per evaluated solution when the number of IoT devices and

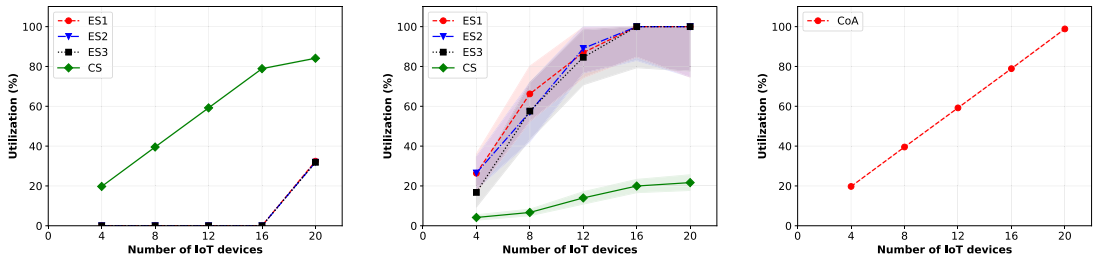


(a) SDN-LB, $d_p^c = 10$ ms (b) RSS, $d_p^c = 10$ ms (c) CoA, $d_p^c = 10$ ms

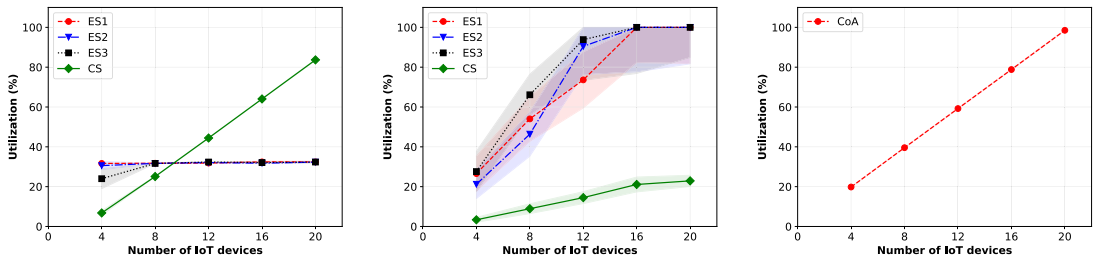


(d) SDN-LB, $d_p^c = 30$ ms (e) RSS, $d_p^c = 30$ ms (f) CoA, $d_p^c = 30$ ms

Figure 4.6: Servers' utilization per approach when frame rate is $r = 10$ fps



(a) SDN-LB, $d_p^c = 10$ ms (b) RSS, $d_p^c = 10$ ms (c) CoA, $d_p^c = 10$ ms



(d) SDN-LB, $d_p^c = 30$ ms (e) RSS, $d_p^c = 30$ ms (f) CoA, $d_p^c = 30$ ms

Figure 4.7: Servers' utilization per approach when frame rate is $r = 15$ fps

communication latency to the cloud server increase. In the plots, ES1, ES2, and ES3 mean the latency at the edge server 1, 2, and 3, respectively, while CS represents the results for the latency at the cloud server. The first trend to be observed is that the SDN-LB proposed solution balances the

workload in the servers in a way that the observed latency in the edge servers is similar (please refer to Figs. 4.4a and c). Besides, the average latency at the edge servers does not change substantially when the number of IoT devices increases. This is because the load posed by the addition of the IoT devices is served by the cloud server. Therefore, the proposed solution also contributes to the fair processing of the video frames from the different flows. Yet, it can be observed that the average latency in the cloud server increases as the communication latency to reach it increases. In the related approach, i.e., RSS solution, the latency observed in the servers is not even, since no balancing mechanism is employed (see Figs. 4.4b and d).

Fig. 4.5 portrays the results of the average latency per server, for the different solutions, when an increased frame per seconds rate is used. The results show a similar trend to the ones discussed above. However, since the frame rate is higher, the latency in all servers increases when a moderate to a high number of IoT nodes are considered (see Fig. 4.5a). Yet, the latency at the cloud and edge servers are similar when a communication latency, from the IoT devices to the cloud server, of 30 ms was considered (see Fig. 4.5c). This is because more frames are offloaded to the cloud server, due to the increased resource demand. In all considered scenarios, the RSS approach presented poor performance due to the lack of load balancing among the edge and cloud servers (see Figs. 4.5b and d).

Figs. 4.6 and 4.7 show the utilization of the servers for different settings of a number of IoT devices, video frame rate, and communication latency from IoT devices to the cloud server. An interesting trend to be observed in the proposed SDN-LB solution is that the utilization in the cloud server is always higher than the utilization of the edge servers (see Figs. 4.6a and d, and Figs. 4.7a and d). This happens because it is faster to process video frames in the cloud even though the communication latency from the IoT devices to the cloud is higher than to the edge servers.

Another interesting trend to observe is that the proposed SDN-LB solution maintains similar utilization at the edge servers, which was around 20% and 30% for low and moderate frame rates (see Fig. 4.6d and Fig. 4.7d). Besides, the edge servers utilization is maintained almost the same even when the workload in the system increases, i.e., the number of IoT devices increases. This is because offloading more video frames to the edge server would increase the experienced latency given their limited processing capability. Hence, the results show that it is preferable to offload the

increased load to the distant cloud server given the considered communication latency cost.

The same desired behavior is not observed in the RSS approach. As shown in Figs. 4.6b and e, and Figs. 4.7b and e, as the utilization in the edge devices increases as the workload increases. This is expected and happens because the RSS approach randomly assigns the IoT flows to the servers without considering the experienced latency or workload on them. When the cloud-only approach is used, no edge server is present in the network to support the processing of edge frames. In other words, all IoT video analytics flows are offloaded to the cloud server. Thus, the utilization in the cloud server increases as more frames are offloaded to it, either by increasing the number of flows, i.e., IoT devices, or by increasing the load at each flow, i.e., offloading frame rate.

4.5.3 Discussions

The obtained results from the proposed modeling and SDN-LB solution provide interesting insights that can be used for the further design of hierarchical edge/cloud solutions for IoT video analytics.

First of all, the load balancing at the servers from the observed latency showed to be a promising approach for supporting latency-sensitive IoT applications, such as video analytics. The advantage of this approach is that servers with different capacities can be added to the network. Besides, the SDN-LB approach shall be promising for scenarios where servers can be added and removed dynamically based on the resource demand. In this case, the server component for IoT video analytics applications can be implemented through microservices or virtual machines in physical server infrastructures, and the virtual machine creation and removal can be done based on the resource demand. In such dynamic implementation, the SDN-LB would seamlessly balance the IoT video analytics flows among the available VMs. Our approach would not directly consider the configurations and resources of the dynamically created VM, but only the estimates of the average latency that frames processed on them would experience.

Besides, results portrayed a maximum utilization for the edge devices in the considered settings (please refer to Figs. 4.6a and c, and Figs. 4.7a and c) when the system's workload increased. The observed utilization on the edge devices depended on the communication latency from the IoT devices to the cloud server, where the higher the latency to the cloud server was, the higher

the utilization on the edge servers was. This is an interesting insight that motivates the design of resource reservation solutions at edge servers based on the communication latency to the cloud server. Hence, during off-peak in the network core and, consequently, low communication latency to the cloud device, part of the edge servers' resource can be allocated to other applications while a given threshold is reserved for the considered IoT latency-sensitive application. Hence, as the core network experiences congestion and increased communication latency to the cloud server, a resource allocation and reservation mechanism would increase the edge servers' resources to be reserved for the IoT application.

4.6 Conclusion

In this chapter, we studied the design of a hierarchical edge/cloud computing environment for IoT video analytics applications. We devised a queuing model to estimate the average latency required to process IoT video frames at each edge and cloud server. Hence, we designed the SDN-LB algorithm, which is a novel algorithm for the load balancing at edge and cloud servers aimed at reducing the latency in IoT video analytics applications. The proposed SDN-LB algorithm relies on an underlying SDN architecture to collect data from programmable switches in a periodic manner. The collected data is then used by the proposed solution to estimate the average latency required to process video frames in each server. Then, the proposed SDN-LB algorithm periodically re-assigns IoT video streaming flows across edge and cloud servers to reduce the latency required to process them. Extensive results showed that the proposed solution efficiently balanced the load at the servers and reduced the latency involved in the task of video frame processing.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we addressed limited computation capabilities of IoT devices to process their produced data and its existing solutions. We discussed about available solutions to cover the computing resources which is needed to process the IoT data i.e., Could servers and Edge servers. Advantages and disadvantages of both of them has been discussed. We also analysed a variety of literature that aims to proposing solutions to improve IoT task offloading and efficiently using IoT, Edge servers, and cloud servers resources. Then, we addressed the deficiencies in these articles, and proposed new methods to improve them.

In this regard, we developed two IoT task offloading architecture for IoT video analytics applications to reduce processing latency as well as minimize the chance of overloaded servers. Our first designed approach, which benefits from SDN, proposed a modeling to calculate the computation latency for streamed video frames at edge servers. Then, we proposed an algorithm that uses a scoring metric to order the edge servers based on their load in a MEC environment to balance the offloaded tasks between the edge servers. The scoring metric is calculated based on the observed incoming and outgoing traffic to each edge servers. The scoring metric is the criteria to re-route the incoming packets to an appropriate edge server. We compared our algorithm with two other related solutions, which demonstrated the out-performance of the proposed algorithm.

In the second proposed scheme, we developed a hierarchical edge/cloud architecture to process

the offloaded tasks from real-time IoT video analytics applications. This work is an extension of the previous work by including cloud servers in the system, when the resources at the edge are not able to address the processing demands of the IoT devices. We used M/M/1 queue to model edge and cloud servers processing latency. M/M/1 queue uses the information gathered by the underlying SDN network to calculate the utilization at each server. Then, by considering the transmission latency at the core and LAN network, in addition to queue delay at the servers, we designed an algorithm that SDN controller periodically redirects the incoming flows from IoT devices to an appropriate server, i.e., selecting an edge or cloud server which leads to a lowest latency. The proposed algorithm is aware of capabilities and existence of more powerful resources in cloud servers compared to edge servers, as well as shorter distance from IoT devices to edge servers which leads to less transmission delay. This knowledge is used by the algorithm to direct the incoming flows to a server with minimized latency. The advantages of the proposed scheme, compared to related solutions, have been demonstrated with numerical evaluations.

5.2 Future Works

The mentioned works can be extended in these directions to address more sophisticated needs of task offloading for IoT video analytics application:

- Develop an adaptive solution for the IoT devices as well, which would enable them to periodically adjust the video frame quality, i.e., resolution, based on observed workload at edge servers aimed at empowering augmented reality applications.

- Since there might be multiple paths to a designated destination, designing an optimal end-to-end routing algorithm can be one of the vital features that an IoT network architecture needs to address.

- Machine learning algorithms have brought many attention in the context of the edge computing (Qu et al. (2021), H. Guo et al. (2020)). Combining SDN controller and machine learning can improve the management capabilities of SDN for incoming flows.

- Explore the resources available in the IoT devices and perform the processing of some frames

locally to further reduce the latency. Optimization algorithms can address the problem of selecting an appropriate place to process streamed video frames with respect to available computing resources and power at IoT devices.

References

Akbar, A., et al. (2021). Sdn-enabled adaptive and reliable communication in iot-fog environment using machine learning and multiobjective optimization. *IEEE Internet of Things J.*, *8*(5), 3057-3065.

Alfakih, T., et al. (2020). Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa. *IEEE Access*, *8*, 54074-54084.

Ali, & Alagan. (2019). A sdn-assisted energy saving scheme for cooperative edge computing networks. In *Proc. of the 2019 ieee global communications conference (globecom)* (p. 1-6).

Altamimi, M., et al. (2015). Energy cost models of smartphones for task offloading to the cloud. *IEEE Transactions on Emerging Topics in Computing*, *3*(3), 384-398.

Ameur, A. B., et al. (2021). On the deployability of augmented reality using embedded edge devices. In *Ieee ccnc* (p. 1-6).

Amjad, A., et al. (2017). Cognitive edge computing based resource allocation framework for internet of things. In *Proc. of the 2017 second international conference on fog and mobile edge computing (fmec)* (p. 194-200).

Apostolo, G. H., et al. (2022). Live video analytics as a service. In *Proc. of the 2nd european workshop on machine learning and systems (euromlsys)* (p. 37–44).

Bastani, F., et al. (2020). MIRIS: Fast object track queries in video. In *Proc. of the 2020 acm sigmod int'l conf. on management of data (sigmod)* (p. 1907–1921).

Belkout, N. E., et al. (2022). A load balancing and routing strategy in fog computing using deep reinforcement learning. In *Proc. of the 2022 international conference on electrical, computer and energy technologies (icecet)* (p. 1-8).

Benzekki, K., et al. (2016). Software-defined networking (sdn): a survey. *Security and Communication Networks*, *9*(18), 5803-5833.

Chen, J., & Jenkins, W. K. (2017). Facial recognition with pca and machine learning methods. In *2017 ieee 60th international midwest symposium on circuits and systems (mwscas)* (p. 973-976).

Chen, M., & Hao, Y. (2018). Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, 587-597.

Chen, Y., et al. (2017). Joint optimization of transmission and processing delay in fog computing access networks. In *Proc. of the 2017 9th international conference on advanced infocomm technology (icait)* (p. 155-158).

Chen, Y., et al. (2023). Dynamic task offloading for digital twin-empowered mobile edge computing via deep reinforcement learning. *China Communications*, 1-12.

Coutinho, R. W. L., & Boukerche, A. (2019). Guidelines for the design of vehicular cloud infrastructures for connected autonomous vehicles. *IEEE Wireless Communications*, *26*(4), 6-11.

Coutinho, R. W. L., & Boukerche, A. (2020). Modeling and analysis of a shared edge caching system for connected cars and industrial iot-based applications. *IEEE Trans. on Industrial Informatics*, *16*(3), 2003-2012.

Coutinho, R. W. L., & Boukerche, A. (2022a). Design of edge computing for 5g-enabled tactile internet-based industrial applications. *IEEE Communications Magazine*, *60*(1), 60-66.

Coutinho, R. W. L., & Boukerche, A. (2022b). Transfer learning for disruptive 5g-enabled industrial internet of things. *IEEE Transactions on Industrial Informatics*, *18*(6), 4000-4007.

Deng, R., et al. (2016). Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. *IEEE Internet of Things Journal*, *3*(6), 1171-1181.

Fernández-Sanjurjo, M., et al. (2021). Real-time multiple object visual tracking for embedded gpu systems. *IEEE Internet of Things Journal*, *8*(11), 9177-9188.

Goudarzi, M., et al. (2021). An application placement technique for concurrent iot applications in edge and fog computing environments. *IEEE Transactions on Mobile Computing*, 1298-1311.

Guo, H., et al. (2020). Toward intelligent task offloading at the edge. *IEEE Network*, *34*(2),

128-134.

Guo, X., et al. (2016). An index based task assignment policy for achieving optimal power-delay tradeoff in edge cloud systems. In *Proc. of the 2016 ieee international conference on communications (icc)* (p. 1-7).

Hamed, M. I., et al. (2017). A new approach for server-based load balancing using software-defined networking. In *Icicis* (p. 30-35).

Hanyao, M., et al. (2021). Edge-assisted online on-device object detection for real-time video analytics. In *Ieee infocom* (p. 1-10).

He, Y., et al. (2018). Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, *67*(1), 44-55.

Holland, O., et al. (2019). The ieee 1918.1 "tactile internet" standards working group and its standards. *Proc. of the IEEE*, *107*(2), 256-279.

Hu, S., & Xiao, Y. (2021). Design of cloud computing task offloading algorithm based on dynamic multi-objective evolution. *Future Generation Computer Systems*, *122*, 144-148.

Kai, C., et al. (2021). Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability. *IEEE Transactions on Cognitive Communications and Networking*, *7*(2), 624-634.

Kang, D., et al. (2019, dec). BlazeIt: Optimizing declarative aggregation and limit queries for neural network-based video analytics. *Proc. VLDB Endow.*, *13*(4), 533–546.

Kaur, S., et al. (2016). Implementation of server load balancing in software defined networking. In *Information systems design and intelligent applications* (pp. 147–157).

Khanna, A., & Kaur, S. (2020). Internet of things (iot), applications and challenges: A comprehensive review. *Wireless Personal Communications*, 1687-1762.

Li, J., et al. (2020a). Energy-aware task offloading in the internet of things. *IEEE Wireless Communications*, *27*(5), 112-117.

Li, J., et al. (2020b). A secured framework for sdn-based edge computing in iot-enabled healthcare system. *IEEE Access*, *8*, 135479-135490.

Li, P., et al. (2022). Deep reinforcement learning for load balancing of edge servers in iov. *Mobile*

Networks and Applications, 1461–1474.

Li, S., et al. (2018). Joint admission control and resource allocation in edge computing for internet of things. *IEEE Network*, *32*(1), 72-79.

Lin, C., et al. (2020). Spatiotemporal congestion-aware path planning toward intelligent transportation systems in software-defined smart city iot. *IEEE Internet of Things Journal*, *7*(9), 8012-8024.

Lin, P., et al. (2021). Resource management for pervasive-edge-computing-assisted wireless VR streaming in industrial internet of things. *IEEE Trans. on Industrial Informatics*, *17*(11), 7607-7617.

Liu, J., et al. (2016). Delay-optimal computation task scheduling for mobile-edge computing systems. In *2016 ieee international symposium on information theory (isit)* (p. 1451-1455).

Liu, J., et al. (2022). Reliability-enhanced task offloading in mobile edge computing environments. *IEEE Internet of Things Journal*, *9*(13), 10382-10396.

Liu, Q., et al. (2018). An edge network orchestrator for mobile augmented reality. In *Ieee infocom* (p. 756-764).

Liu, R. W., et al. (2022). Deep network-enabled haze visibility enhancement for visual iot-driven intelligent transportation systems. *IEEE Transactions on Industrial Informatics*, 1-11.

Liu, W., et al. (2016). SSD: single shot multibox detector. In *Eccv* (pp. 21–37).

Liu, X., et al. (2020). Abnormal traffic congestion recognition based on video analysis. In *Proc. of the 2020 ieee conference on multimedia information processing and retrieval (mipr)* (p. 39-42).

Lyu, X., et al. (2017). Multiuser joint task offloading and resource optimization in proximate clouds. *IEEE Transactions on Vehicular Technology*, *66*(4), 3435-3447.

Meng, T., et al. (2018). A secure and cost-efficient offloading policy for mobile cloud computing against timing attacks. *Pervasive and Mobile Computing*, *45*, 4-18.

Misra, S., & Saha, N. (2019). Detour: Dynamic task offloading in software-defined fog for iot applications. *IEEE JSAC*, *37*(5), 1159-1166.

Mohamed, R., & Zemouri, S. (2022). Towards a cloud-native 5g service chaining for iot and video analytics in smart campus. In *Proc. of the 5th conf. on cloud and internet of things (ciot)*

(p. 186-188).

Nogueira, V., et al. (2019). Retailnet: A deep learning approach for people counting and hot spots detection in retail stores. In *Proc. of the 2019 32nd sibgrapi conference on graphics, patterns and images (sibgrapi)* (p. 155-162).

Pang, S., et al. (2021). A smart network resource management system for high mobility edge computing in 5G internet of vehicles. *IEEE Trans. on Network Science and Engineering*, 1-13.

Poddar, R., et al. (2020). Visor: Privacy-Preserving video analytics as a cloud service. In *Proc. of the 29th usenix security symposium* (pp. 1039–1056).

Qian, W., & Coutinho, R. W. L. (2021). On the design of edge-assisted mobile iot augmented and mixed reality applications. In *Acm q2swinet* (p. 131–136).

Qu, G., et al. (2021). Dmro: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing. *IEEE Transactions on Network and Service Management*, *18*(3), 3448-3459.

Redmon, J., et al. (2016). You only look once: Unified, real-time object detection. In *Ieee cvpr.*

Roy, D. G., et al. (2017). Application-aware cloudlet selection for computation offloading in multi-cloudlet environment. *The Journal of Supercomputing*.

Singh, N., et al. (2022). Iot enabled helmet to safeguard the health of mine workers. *Computer Communications*, *193*, 1-9.

Sun, X., & Ansari, N. (2016). Edgeiot: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, *54*(12), 22-29.

Tong, L., et al. (2016). A hierarchical edge cloud architecture for mobile computing. In *Ieee infocom 2016 - the 35th annual ieee international conference on computer communications* (p. 1-9).

Varghese, B., et al. (2016). Challenges and opportunities in edge computing. In *Proc. of the 2016 ieee international conference on smart cloud (smartcloud)* (p. 20-26).

Wan, S., et al. (2021). An intelligent video analysis method for abnormal event detection in intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, *22*(7), 4487-4495.

Wei, X., et al. (2017). Mvr: An architecture for computation offloading in mobile edge computing. In *Proc. of the 2017 ieee international conference on edge computing (edge)* (p. 232-235).

Wu, H., et al. (2021). Eedto: An energy-efficient dynamic task offloading algorithm for blockchain-enabled iot-edge-cloud orchestrated computing. *IEEE Internet of Things Journal*, *8*(4), 2163-2176.

Xu, X., et al. (2019). A computation offloading method over big data for iot-enabled cloud-edge computing. *Future Generation Computer Systems*, *95*, 522-533.

Yount, Z. F., et al. (2022). Route learning with augmented reality navigation aids. *Transportation Research Part F: Traffic Psychology and Behaviour*, *88*, 132-140.

Yu, S., et al. (2020). Intelligent edge: Leveraging deep imitation learning for mobile edge computation offloading. *IEEE Wireless Communications*, *27*(1), 92-99.

Yuchong, L., et al. (2019). Task scheduling in mobile edge computing with stochastic requests and m/m/1 servers. In *Proc. of the 2019 ieee 21st international conference on high performance computing and communications* (p. 2379-2382).

Zeng, X., et al. (2020). Distream: Scaling live video analytics with workload-adaptive distributed edge intelligence. In *Proc. of the 18th conference on embedded networked sensor systems (sensys)* (p. 409–421).

Zhang, K., et al. (2016). Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access*, *4*, 5896-5907.

Zhang, Y., et al. (2021). High-performance isolation computing technology for smart iot healthcare in cloud environments. *IEEE Internet of Things Journal*, *8*(23), 16872-16879.

Zhu, W., et al. (2022). Recognition of interactive human groups from mobile sensing data. *Computer Communications*, *191*, 208-216.