Software-Defined Computational Offloading for Mobile Edge Computing

by

Nitesh Krishna

Thesis submitted in partial fulfillment of the requirements
For the Master of Applied Science degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Computational offloading advances the deployment of Mobile Edge Computing (MEC) in the next generation communication networks. However, the distributed nature of the mobile users and the complex applications make it challenging to schedule the tasks reasonably among multiple devices. Therefore, by leveraging the idea of Software-Defined Networking (SDN) and Service Composition (SC), we propose a Software-Defined Service Composition model (SDSC). In this model, the SDSC controller is deployed at the edge of the network and composes service in a centralized manner to reduce the latency of the task execution and the traffic on the access links by satisfying the user-specific requirement. We formulate the low latency service composition as a Constraint Satisfaction Problem (CSP) to make it a user-centric approach. With the advent of the SDN, the global view and the control of the entire network are made available to the network controller which is further leveraged by our SDSC approach.

Furthermore, the service discovery and the offloading of tasks are designed for MEC environment so that the users can have a complex and robust system. Moreover, this approach performs the task execution in a distributed manner. We also define the QoS model which provides the composition rule that forms the best possible service composition at the time of need.

Moreover, we have extended our SDSC model to involve the constant mobility of the mobile devices. To solve the mobility issue, we propose a mobility model and a mobility-aware QoS approach enabled in the SDSC model. The experimental simulation results demonstrate that our approach can obtain better performance than the energy saving greedy algorithm and the random offloading approach in a mobile environment.

Acknowledgements

My most sincere gratitude to Prof. Amiya Nayak for providing me guidance, support, advice, and ideas throughout my thesis work. His guidance and expertise have been invaluable while demanding the most out of me academically and spurring my personal growth. I consider myself lucky to have received his esteemed advice as well as his empathy regarding every situation throughout my research period. I wholeheartedly thank him for his precious supervision.

My sincere appreciations to Mr. Pradeep Srivastava and his entire family for their unconditional love and support during my stay here. I would also like to acknowledge Mr. Ahmad Helmy for his valuable suggestions and guidance. His expertise in the field of my research topic, have greatly influenced my work. Further, a special word of thanks to my parents, family, and friends for keeping my motivation high and supporting me throughout my thesis.

In the end, I would like to thank all my teachers and mentors who have been instrumental in making me capable of pursuing Masters degree.

Contents

List of Tables

List of Figures

List of Abbreviations

AP	Access Point
API	Application Program Interface
CM	Composition Manager
COA	Conditioned Optimal Algorithm
CPU	Central Processing Unit
CSO	Cloud Service Operator
CSP	Constraint Satisfaction Problem
DSOE	Distributed Service Orchestration Engine
EGA	Energy-Saving Greedy Algorithm
EO	Edge Orchestrator
ES	Edge Server
ESO	Edge Service Owner
FME	Follow Me Edge
IoT	Internet of Things
MCC	Mobile Cloud Computing
MEC	Mobile Edge Computing
MECS	Mobile Edge Computing Server

MNO	Mobile Network Operator
OTT	Over The Top
QOS	Quality of Service
RAN	Radio Access Network
REM	Radio Environment Map
SDCOM	Software-Defined Co-operative Offloading
SDN	Software-Defined Networking
SDR	Semidefinite Relaxation
SDSC	Software-Defined Service Composition
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOM	Service Oriented Middleware
SP	Service Provider
SR	Service Requester

Chapter 1

Introduction

Over the years, technology has evolved and became an integral part of human society. Technology has played its role by providing a sustainable technical solution to the social problems and has made the community more connected and productive. In addition to this, all the physical devices are continuously getting connected to the internet making the internet much more extensive and complex. Therefore, this advancement is causing the consumer demand for networked services to integrate into their lifestyle seamlessly. Hence, driving the networking and computation technology into day to day objects and making the concept Internet of Things (IoT) a reality. IoT is a concept in which all the physical devices with or without computation power connected over a network where devices can send, receive data and can communicate with each other to achieve some common goals, making the technology more fascinating. Mobile Edge Computing (MEC) is the concept where computation of the task happens at the edge of the network. The edge of the network can be the base stations, access points, etc. Therefore, many researchers consider MEC as the backbone of IoT.

With the evolution of mobile equipment such as smartphones and laptops, the mobile applications have also evolved tremendously regarding their complexity which leads to the

higher requirement of computational power. The mobile devices are both the consumer as well as the producer of data, and due to the limited resources, it is hard to perform all the tasks on the mobile device itself. This constraint makes the computational offloading a necessity. However, the recent advancement in the cloud computing provides on-demand access to the shared pool of computing resources like the computer network servers, storage, and computation services. It is easier to have all the tasks offloaded to the cloud server which then gets executed by the cloud, and the result is sent back to the mobile devices.

In this thesis, we propose an architecture for dynamic service composition with the use of Software-Defined Networking (SDN) framework. The SDN technology is a novel approach in networking paradigm which facilitates the network management and enables the programmable network configurations for better performance. The service is composed by the SDN controller by solving service selection problem as a Constraint Satisfaction Problem (CSP). A CSP is a mathematical problem defined as a set of objects whose state must satisfy some constraints. In the proposed architecture, the centralized controller composes the service based on the task requested by the mobile device. For example, a complex application running on a smartphone requires a task to be executed by it and the device itself is insufficient to fill out the task. Therefore, it must offload some or most of the computation task to the cloud or the peer devices. It also requires the centralized algorithm to perform the scheduling of the work.

By leveraging the SDN technology, Cui et al.[13] proposed a software-defined cooperative offloading model. In this model, the central controller schedules the tasks based on the request of the mobile devices. However, the offloading decision is based on the energy consumption. We make use of the same centralized SDN model and provide a new framework for the SDN-based service composition for computational offloading.

1.1 Motivation

We focus on the parameters to devise an effective approach for computation offloading. It is motivated by the increase in demand for highly sophisticated mobile applications with low latency execution requests such as image processing and advanced gaming applications. However, these applications require high computation and storage capacity. Therefore, it is hard to achieve the execution of the task by the device alone due to resource constraints. In addition to this, these days mobile devices are equipped with numerous advance network technologies like Wi-Fi, Bluetooth, and Infrared connectivity. Therefore, with the increase in the number of mobile devices getting connected to the internet and keeping the advancement in connectivity features and above parameter, we devise an efficient solution for task execution at the user's vicinity with the help of peer devices.

Consider an example of the smart home concept where every device is connected to the internet. A highly sophisticated application like augmented reality application, gaming or image processing application running on one of the devices, requires high computation capability to execute the task. For a mobile device to complete the overall execution by itself will not be a feasible solution due to resource constraint. On the other hand, the peer or nearby devices can help to achieve some part of the task if they are idle. In such scenarios, there is a need for a centralized architecture to schedules the job to the peer devices so that overall goal of the application is achieved.

Recent advancements in SDN such as logically centralized controller over a distributed resource of mobile devices has motivated us to consider the SDN-based centralized architecture for computational offloading in MEC. The controller does the scheduling of the tasks based on the user requirement of computation and external traffic usage.

We all know that the execution of the task may require execution of several subtasks which burdens the requester device with the re-initiation process of offloading. Therefore, this issue has motivated us to work on a centralized service composition approach using a controller to reduce the overhead on the requester device. We first discuss the centralized service composition model. We further propose an SDN controller-based service composition framework and management of duty. The SDN controller initiates the process of composing a service after receiving the composition request from the service requester node. However, the service composition depends on different constraint which is solved by the controller as a CSP.

1.2 Contributions

This research work provides a service composition approach to achieve an efficient computational offloading in MEC environment using aspects of SDN.

- We propose an SDN-based computational offloading approach for MEC. Our approach uses service composition technique to compose the service from the services available in the vicinity of the user. We use CSP for service selection and solve it using backtracking and forward checking algorithm. Our approach uses SDN controller for the overall process. This ensures that the process is centralized and reduces the overhead on the requester device. Also, with the approach, we ensure that the composed service is the best possible service composition available in the network. Our algorithm ensures that the users requirement has been satisfied while selecting the services.

- We propose a mobility-aware mechanism for service composition in MEC using SDN. Our approach uses mobility pattern to predict user's future location, and the

SDN controller composes service at the next location. We employ the concept of quality of service at the future location to determine the best possible services for service composition.

1.3 Organization

This thesis is organized as follow:

- Chapter 2 provides work related to the MEC, task offloading, SDN, and service composition. We have provided the applications and frameworks where these methods have been deployed.

- Chapter 3 presents the SDN-based service composition protocol. We have discuss the existing SDN framework for computational offloading and consider it as a benchmark for our research. Then, we investigate the centralized service composition protocol and proposed a controller-based service composition using SDN framework. We use CSP to solve the problem of selecting the nodes for service composition based on the user constraints, and QoS service ranking model is used to fetch the best possible service composition.

- Chapter 4 presents the mobility-aware approach which further enhances the SDN-based service composition protocol. For this, we have discussed how SDN controller can get the future location of the user at the time of service request. We use trajectory information to predict the future location of the user and develop the service ranking for selecting the node which involve the response time of service at the next position.

- Chapter 5 summarizes the contributions and future work.

Chapter 2

Background and Related Work

With the increase in IoT solutions and highly complex applications, there has been concern from the consumer for latency and data privacy. Since then, there is a desire for the computation of the task at the edge of the network within the proximity of the user. Therefore, MEC has been the topic of interest in the past few years. Moreover, with the increase of interactive mobile application, MEC has been considered by academia as the backbone of IoT. The recent advancement in mobile devices concerning their computation capability and the storage have brought the computational resources closer to the user, making edge computing [2] [40] a reality, which is also similar to Fog computing [7] [38]. These mobile devices have evolved with the advanced communication mechanisms like Wi-Fi, Bluetooth, and Infrared which makes it easier for the devices to share data with each other and form networks of their own.

The diverse nature of the interactive application and ultra-low latency user requirement in edge computing environment has made the cloud-based mobile computing techniques such as cloud offloading [31] not applicable for this new paradigm as a sole solution. However, the involvement of cloud in the edge computing environment cannot be removed but can be reduced to an extent to achieve low latency [55]. The trade-off between the

latency and computational offloading is studied while building decision-making algorithm to schedule the computation of the task on the local or remote devices. Therefore, many research is in progress to find the efficient solution to this problem.

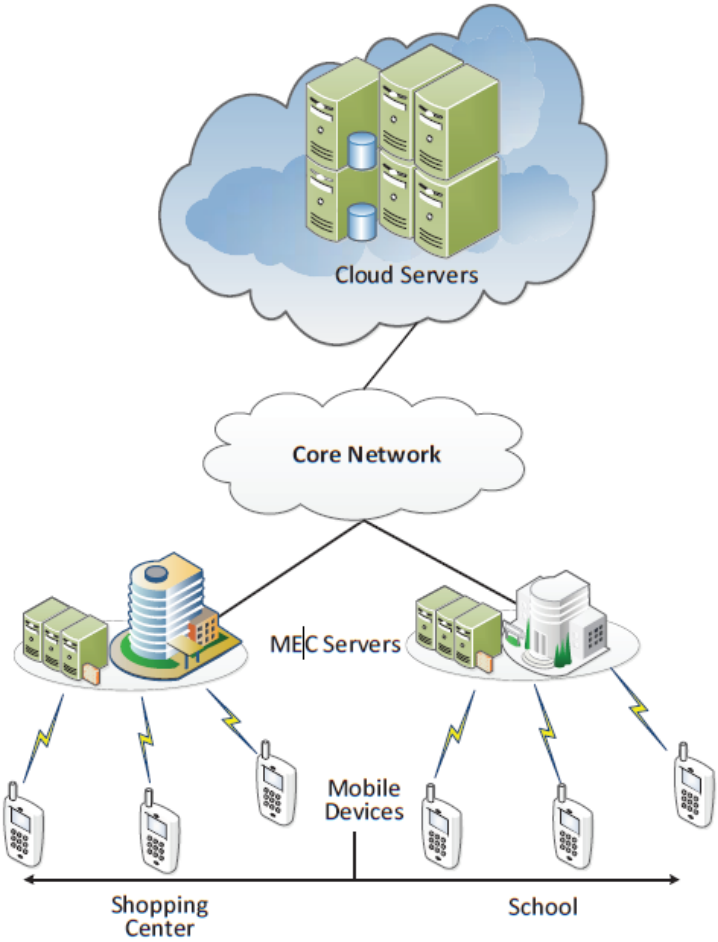

Figure 2.1: MEC Architecture [2]

Figure 2.1 introduces to the architecture of the MEC application. The design described here is divided mainly into three components. The first component consists of edge devices which include all types of devices like smartphones, embedded networked appliances [43], IoT devices [55]. The second part is a core network which acts as the backbone of the network. The third component includes mobile edge cloud, cloudlet

[52] or Mobile Edge Computing Servers (MECS) which are restricted cloud resources deployed at the base station in the user's proximity. MECS has the responsibility to maintain the traffic flow and filter the traffic.

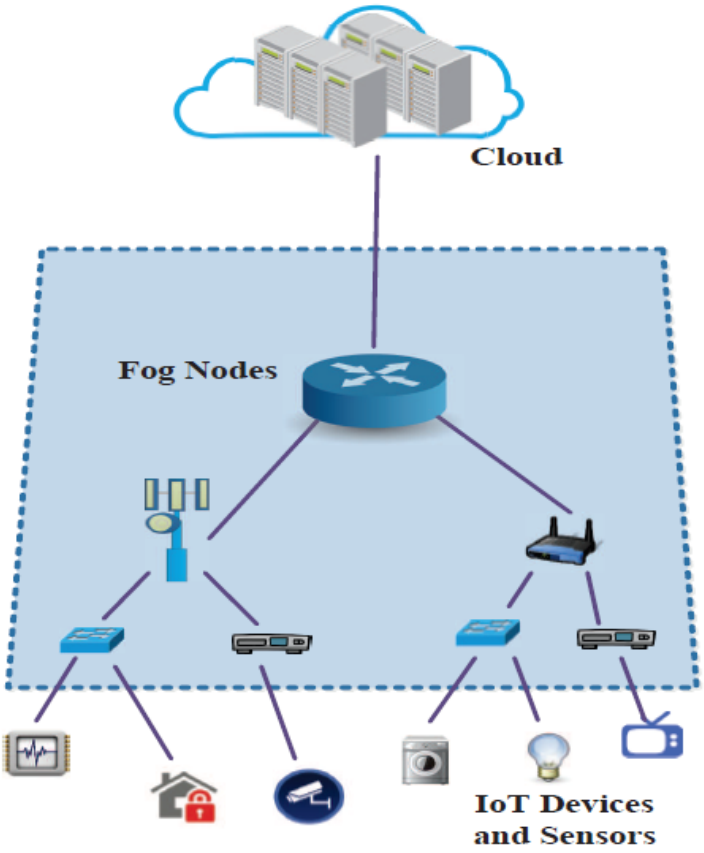

Figure 2.2: Fog Computing Environment [41]

Figure 2.2 is depicting the Fog computing environment. Fog computing is a distributed computing paradigm which acts as a middle layer between the mobile nodes and the cloud service provider. Services provided by the Fog computing such as computation, networking, and storage allow users to get cloud-like service in their proximity. Fog nodes are one of the fundamental requirement to achieve the fog computing environment. There are multiple ways to establish a Fog environment. Fog servers are small cloud-like servers which are distributed in nature and deployed at frequent places like

railway stations, shopping center. The network devices such as routers, switches, set-top box create Fog nodes using virtualization technique and offer services at edges of the network. We summarize the Fog computing as a virtualize cloud-like device deployed in the user's vicinity and therefore it is interpreted as "cloud close to the ground". However, according to the existing research work on MEC and service offloading, it divides further into two categories:

- Traffic offloading: Due to increasing demand for low latency multimedia services and reducing bandwidth on the cellular data, it is required to have some of the cloud data to offload to the Cloudlets [52] or to Mobile Cloud [23] near to the user. For example, if a user has streamed a video, it should be cached to the mobile cloud for further streaming by other users. This idea has been implemented in a metropolitan city as suggested by Lee et al. [34].

- Computational offloading: In computational offloading, the tasks which require high computational resources are offloaded from the device to the near computer or server which can perform the computation of the task. Numerous attempt has been made to offload the work of the mobile devices to a remote computer for computational processing. We have focused our thesis on this type of offloading.

In the following two sections, we will discuss both offloading approaches with their applications. However, we discuss computational offloading more as it is our primary focus in the thesis.

2.1 Traffic Offloading in MEC

Traffic offloading is one of the approaches which has been taken into consideration to reduce the bandwidth congestion and burden on the cloud. However, due to an exponen-

tial increase in user data one such approach is required to provide consumers a fast and quality services, making smart homes and cities a reality. Therefore, researchers have proposed many concepts for traffic offloading, and one such approach is Follow Me Edge (FME) [59]. The idea behind FME is similar to Follow Me Cloud[60].

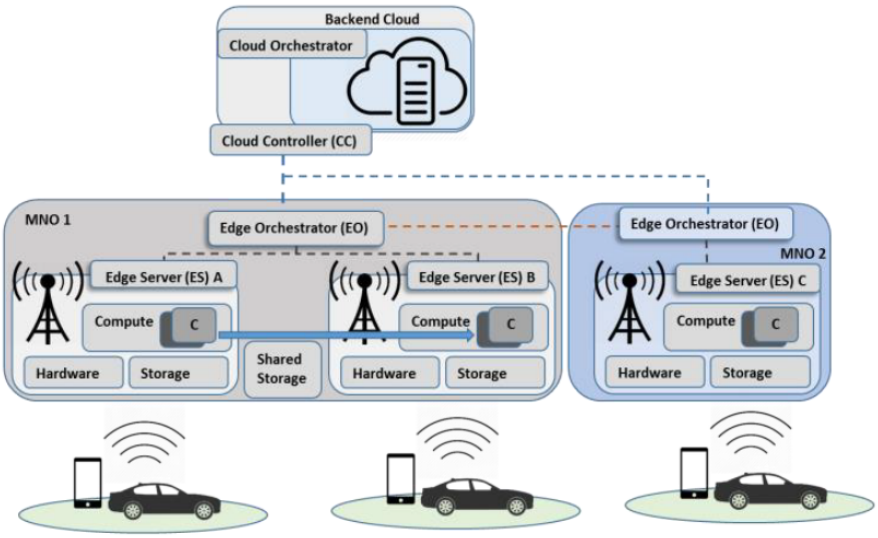

Figure 2.3: The Proposed MEC Architecture for FME [59]

The proposed FME architecture is based on two-stage principle process as shown in Figure 2.3. First, the content provider which can be the third party Over The Top (OTT) service provider approaches Cloud Service Provider (CSP) through Application Program Interface (API) to get access to implement its application. Its orchestrator manages all the services and infrastructure offered by CSP. To maintain the service level agreement between the OTT and Mobile Network Operator (MNO), cloud controller is used. An edge server is deployed by the MNO to provide the services to the edge nodes such as smartphones. An Edge Orchestrator (EO) maintains all the edge related services. Figure 2.3 provides the MEC architecture for FME and depicts the flow of the services.

Initially, when a edge device requests the service, it is offered by the CSP. The cloud provided service may have a delay and low quality. To overcome this issue EO may

initiate the service to replicate on the ES. The ES can reproduce all the service or the part of it based on the requirement. After the replication process, the edge device gets high-quality service, and QoS is achieved because the content is just one hop away from the edge device. Since the Service-Level Agreement (SLA) is present between the OTT and MNO, the ES can take measures on pre-agreed negotiations. Furthermore, the user devices are not always static and they tend to move with the user. Therefore, the concept of FME is to follow the user device so that the ES can leverage the SLA. Based on the MEC's active device user tracking, the EO may predict the next location of the user. Hence, it can replicate the content to the nearest ES to offer seamless service to the user.

2.2 Computational Offloading in MEC

Computational offloading in MEC has been gaining interest since the evolution of the complex applications and ultra-low latency consumer demands. IoT [14] concept, which has attracted many researchers in the recent years, have multiple devices which are scattered and connected to each other over the internet resulting an enormous amount of data to be executed by the IoT devices [55] at the edge of the network. For example, a highly sophisticated application running on the IoT device requires a complex computation. The device itself is unable to perform as it has resource constraints, desires to offload some of its tasks to other devices in its proximity or the cloud for execution, it is termed as computational offloading. Moreover, the production as well as the consumption of data, both are at the edge, makes it highly distributed.

Thinh et al. [16] presented an adaptive computation scaling and task offloading model for MEC. The model consists of the mobile devices capable of offloading the computational work to Access Point (AP) with computational capabilities as shown in Figure 2.4. Firstly, they derive the optimal CPU frequency which is required to adopt

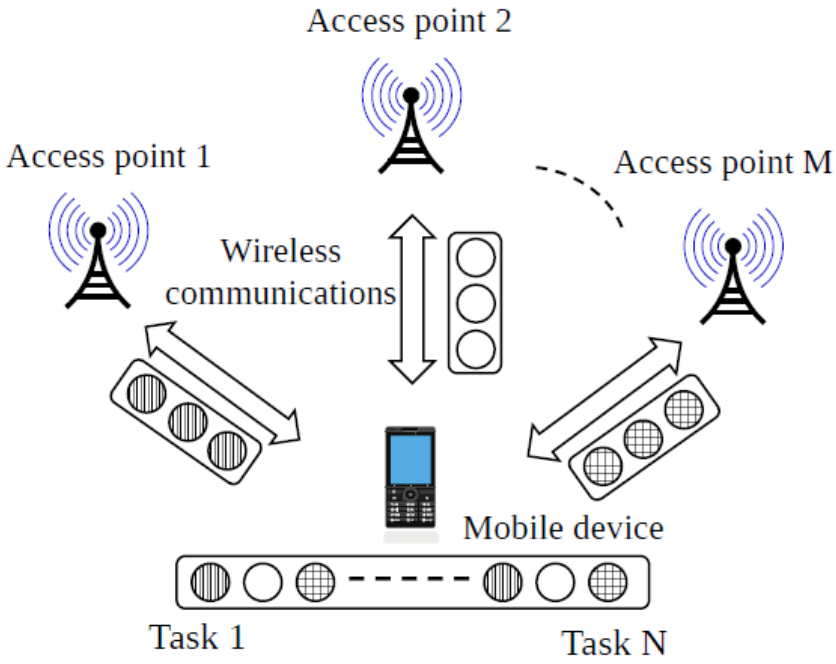

Figure 2.4: Offloading Framework with Multiple AP [16]

the given task allocation decision. The primary objective defined by the authors is to minimize the task execution latency and reduce the mobile device energy consumption. The authors have discussed an exhaustive search approach to find the optimal solution. However, in worst case scenario the procedure iterates through all the solution to find the feasible solution which makes it less efficient. Thus, it is only used to compare with other lower complexity methods. The authors then proposed a Semi-Definite Relaxation (SDR) based approach, which begins by transforming the trade-off equation to an equivalent homogeneous Quadratically Constrained Quadratic Program (QCQP). It is further solved by the convex solvers and without the ranking constraint.

Similarly, Yang et al. [37] have proposed a game theory approach for computational offloading in edge computing. They consider the environment between the Cloud Service Operator (CSO) and the Edge Service Owner (ESO) where they formulate the interaction as a Stackelberg game, which enables the CSO to allocate the computation based on

ESO's valuation. The authors have designed two computational offloading algorithms and quantizes the efficiency of both algorithms as little delay and reduced complexity.

Koya et al. [51] proposed radio environment-aware task allocation method. They consider multiple MEC environment and the work is allocated simultaneously to the server. The authors formulate wireless connectivity as the function of the distance from a given MEC server. The authors have presented two methods Radio Environment Map (REM) and distance information assisted method to predict the wireless connectivity correctly. Based on the connectivity and the distance information, the expected value for the amount of collectible calculation results per unit time is maximized. The proposed solution is one such approach where researchers consider radio environment as a constraint to solve the offloading problem.

Yuan et al. [67] proposed a solution for mobile code offloading. They offer an efficient code partitioning algorithm based on the observation that when the device offloads a method, the subsequent invocation will offload with the high chance. The clustering offloading property of the call graph has motivated this design. This approach considers partitioning of the overall application and the methods of the application. Since complex application usually consists of many composable components and execution of all is not possible by the mobile device alone due to the resource constraint. Thus, the methods itself is offloaded by the devices.

Xu Chan et al. [11] adopted a game theoretic approach to achieve the efficient computation offloading in a distributed manner. Thus, the authors formulate the distributed computation offloading decision-making problem among the mobile device users as a multi-user computation offloading game. Further, the authors have designed a distributed computation offloading algorithm that can achieve the Nash equilibrium. It is done by deriving the upper bound of the convergence time and quantify its efficiency

ratio over the centralized optimal solutions regarding the essential performance matrices. The authors have further extended the work in a multi-channel wireless contention environment.

In recent advancement in the Mobile Cloud Computing (MCC), researchers have proposed several approaches. MAUI [12] and ThinkAir [30] provide method-level computation offloading and do not require exceptional support from the operating system. They still need the programmer to access the source code and partition the application manually. SociableSense [48] shows how the society related application can benefit from the cloud. These schemes focus on how to partition the tasks statically or dynamically, realizing method-level or application-level migration, and when to offload applications from a device perspective.

However, the research on optimal policies for scheduling multiple tasks is limited. Therefore, the primary challenges are still how to gather the distributed information about the mobile devices and schedule tasks to the appropriate devices or the cloud. Many discussed solutions are infrastructure based as they evidently offload either to the access point or the cloud. They lack in collaborative approach where multiple tasks can be simultaneously being offloaded to the cloud and can be executed with the help of peer devices. However, departing from the existing solutions, our schemes not only focus on minimizing the latency but also guarantees to meet the user requirements.

2.3 Software-Defined Networking

Today's networking era is more hardware-centric where various chips and purpose-driven Application Specific Integrated Circuits (ASICs) are used to satisfy throughput need of the network. However, the current networking system has some severe limitations like manageability, flexibility, and extensibility. Network operators are restricted to use pre-

defined commands and configuration based on embedded operating system and firmware, even though it will be easier to use more configurations and protocols based on the specfic network. Therefore, it would be more efficient to program network controls in a sense to make it more responsive and flexible. As to overcome these limitations, the SDN has been proposed and achieves substantial attention from both academia and the industry.

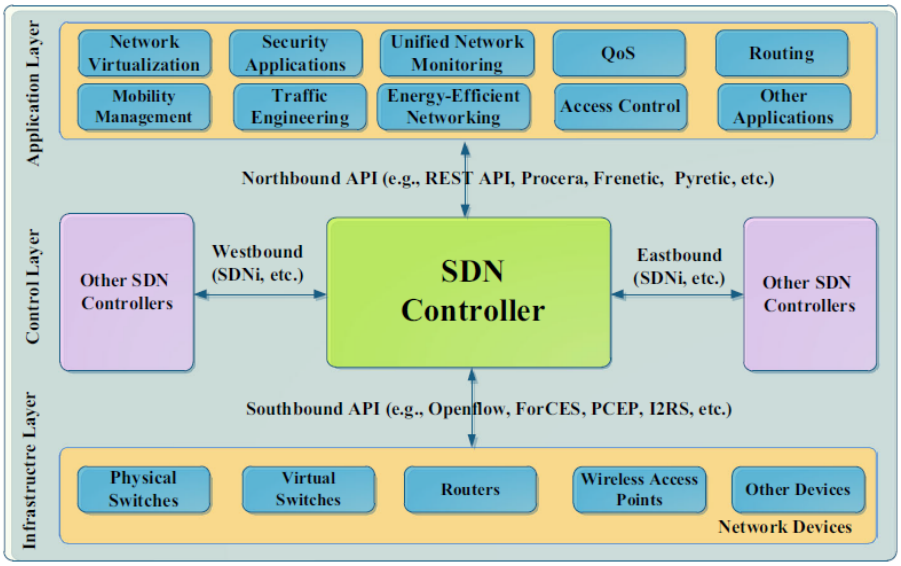

Figure 2.5: Overview of SDN Architecture [46]

SDN [28] [42] [53] has emerged as a new network architecture. The main advantage of the SDN is that it decouples the control plane from the forwarding plane. Moreover, SDN provides simplified network management and it promotes the innovation through the programmability of the network. Figure 2.5 depicts the SDN architecture illustrating the separation between the application, control and data plane. The SDN provides easy deployment for innovative ideas and services. It dramatically reduces the cost through programmable interface in controller like OpenFlow [57], ForCES [17], and PCEP [33]. The SDN framework provides a centralized controller which controls the data path element originated from the different vendor devices and are independent of the network technology. The main feature of the SDN controller is to maintain the over-all network-

wide view of the data path element and links that connect them. The controller makes use of the updated network-wide view of the data path to provide intelligent network performance and resource utilization.

Southbound API is supported by the controller to communicate with the network devices. At this level, the network device is required to support the API. Northbound API is for the communication between the applications and the controller, where the applications used to enforce their policy to the infrastructure device through the controller. Similarly, the interaction between the different SDN controllers takes place using eastbound/westbound API [46].

2.3.1 Application Layer

The logically decoupled control plane has made it easier to make the changes in network policies and information collection. Thus, the SDN technique can be exploited to achieve different business goals. Application layer consists of various business applications which categorize into three types such as special purpose, network application, and security.

One such special purpose application is the network management. In [24], the authors propose an Aster*x, which is a porotype distributed load balancer. They have used OpenFlow to measure the state of the network and update the flow table accordingly. Similarly, in [1] the authors utilize the control information collected by the control plane to reduce the delay and packet loss which in turn improves the network utilization.

Minlan et al. proposed the OpenSketch [66] which is a software-defined traffic measurement architecture. It is used to support more dynamic and customized traffic measurement with accuracy. It is achieved by decoupling of the measurement data plane from the control plane, where data plane supports many analysis tasks, and the control plane allocates and configures different measurement functions.

Academia widely researches SDN for network applications such as network monitoring, bandwidth optimization, redundant traffic removal, etc. The authors in [26] provide trimming approach to remove unnecessary headers, lower in the protocol stack called Scissor. They introduced the notion of Flow ID, where all the packets are identified based on its Flow ID belonging to its flow. This approach reduces the latency of network traffic in data centers. Similarly, for optical networks, the SDN controller is integrated with the optical switching to explore the application and work control [63]. It further optimizes the application performance and reduces the control overhead. This research shows that the SDN with an optical network can provide the innovative approach to achieve long-term goals.

2.3.2 Control Layer

Control plane is the main advantage of the SDN architecture. It is separated from the application and data plane hence provide better governance of the network. In SDN, the network management is logically centralized in the control plane consisting of one or more controller which may host many control applications. Since the controller provides the programmatic interface to the entire network, many applications can perform the management tasks and offer new functionalities to the controller. The controller communicates to the application layer through the Northbound API and to data plane through the Southbound API. Research related aspect involved in the controller mainly falls into the following categories.

- The user-defined applications are designed and deployed at the controller to efficiently manage and operate the network.

- The architecture of controller profoundly affects the performance of SDN. Many different designs have been proposed recently such as the multi-core controller, the logically centralized controller, and the wholly distributed controller.
- The single controller exhibits the limitations of performance and scalability. Meanwhile, the placement problem of distributed controllers also affects the network performance.
- The mainstream interfaces associated with the controller are essential components to connect the users and network devices to realize the SDN.
- The controller is the core of SDN. If it is attacked and become undependable, the entire network will become useless.

Table 2.1: A Brief Comparison Between Various Controllers

Controller	Category	Southbound interface	Platform	Development team
ONOS	Logically centralized	OpenFlow/Others	Win/Mac/Linux	ON.Lab
OpenDaylight	Logically centralized	OpenFlow/Others	Win/Mac/Linux	Linux foundation
Beacon	Single controller	OpenFlow	Win/Mac/Linux	Stanford
Floodlight	Logically centralized	OpenFlow	Win/Mac/Linux	Big Switch Networks
NOX/POX	Logically centralized	OpenFlow	Linux	Nicira Networks
Ryu Controller	Logically centralized	OpenFlow/Others	Win/Mac/Linux	NTT Labs
McNettle	Single controller	OpenFlow	Linux	Yale university

Furthermore, we discuss the various aspects of the control plane as the controller is the key feature of the SDN infrastructure. Table 2.1 provides a brief comparison between various controllers currently available in the market. The SDN controller provides

the programming interface to the entire network. As proposed by the authors in [9], a single controller can manage up to 10,000 machines which may be useful for small internet topologies but indeed cannot manage large topologies due to resource constraints. To improve scalability of the SDN structure, researchers have proposed many control architectures. The design falls into two categories, the single control planes and the multiple control planes. For single control plane, some researchers believe that one controller would be enough for the entire network and the problem is how to enhance its performance. For example, Maestro [45] is one such approach which deals with the single controller. The second category deals with the multi-controller approach where multiple controllers collaborate with each other to manage the entire network such has HyperFlow [61].

2.3.2.1 The Multi-Core Controller

It is true that if SDN ever had to become the dominant network architecture, it must be able to withstand the high flow demand in the vast networks. In such system, the SDN's main issue is to provide optimal routing for such flows in time. If the SDN controller does not have enough capacity to handle the flow requests of the system, it will become a bottleneck of the entire system. Many researchers have proposed work related to the issues such as Beacon [18]. Beacon utilizes multiple core controller to achieve the high efficiency. The beacon framework provides control over the network devices using the OpenFlow protocol. It also has a set of built-in applications for the control function and can control over 12 million flow requests per second with 12 cores.

2.3.2.2 Logically Centralized Controller

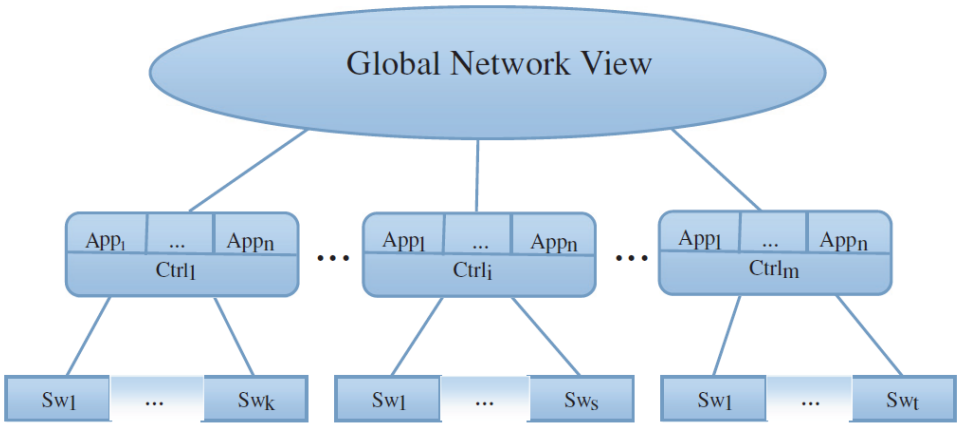

Figure 2.6: An Overview of Logically Centralized but Distributed Controller Architecture [65]

From previous discussions, we already know that the all-knowing controller is a vital attribute of SDN to manage the entire network such as NOX [20]. The performance of the multi-core controller is much better than the traditional single controller, but it has some severe disadvantages such as single point failure which can be fatal to the entire network. To solve the issue, the distributed but logically centralized architecture is introduced such as ONOS [6]. In this architecture, the distributed controllers share information with each other to build the entire network view. Figure 2.6 describes the overview of the logically centralized architecture. We can see that the controllers can share the network-wide view.

However, to maintain the network view in traditional network, the controllers needs to synchronize with each other and perform every function such as routing mechanism, state distribution and failure mechanism on their own. The lack of one standard control plane significantly hinders the flexibility and the reliability of the network. Onix [29] was proposed to solve this issue. It is a distributed system which provides generic

distributed state management API to the programmer to program the control logic. In the context of Onix, the control plane operates the entire network globally making it logically centralized with the distributed controller.

In practical networks, the network operators can deploy many controllers on demand. By propagating events that affects the state of the controllers, HyperFlow [61] can enable all the controllers to achieve the network-wide view by passively synchronizing the network-wide views of the OpenFlow controllers. The HyperFlow provides the logically centralized control, which consists of many distributed controller and provides scalability. HyperFlow is implemented as an application in NOX [20]. As each controller knows the global view of the network, it is easy for each controller to decide on flow rule thus minimizing the response time of the network.

Similarly, Disco [47] which is another open and extensible SDN control plane that copes with the distributed and heterogeneous characteristics of vast area network and modern overlay network. It behaves as a logically centralized controller, each of it distributed controllers manages its network domain and synchronizes with other controllers through lightweight network channels to maintain end to end service. They can adopt changing topologies and is resilient to attack and disruption.

2.3.2.3 Distributed Controller

As discussed previously, the control plane is an advantage of the SDN architecture where the centralized controller can manage the network resources and take the decision on flow rules based on the global network view. However, this centralized structure can have some side-effect as well such as inconsistency among individual controllers contributing to the network in ever-changing network topologies and it may lead to increase in the workload. Moreover, Levin et al. [35] found that the inconsistency degrades the performance of any

application. Therefore, the distributed controller became inevitable as the centralized architecture is inadequate in a functional network.

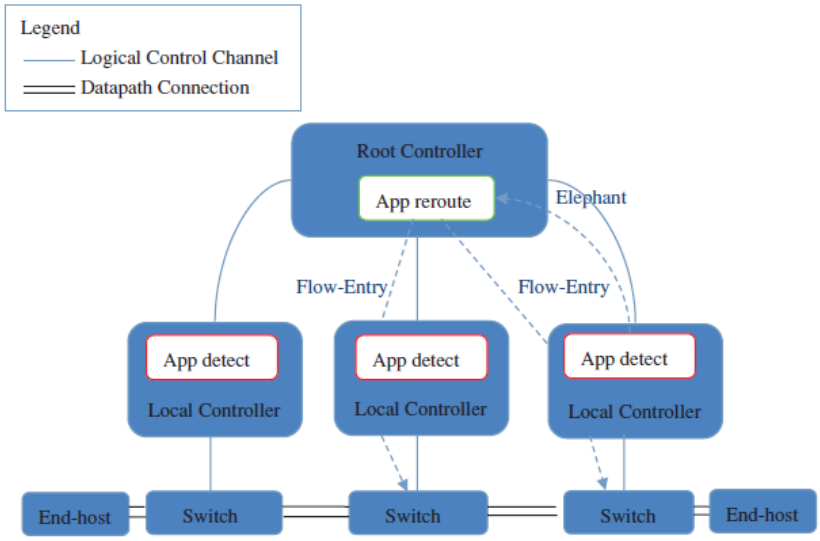

Figure 2.7: An Overview of Distributed Controller Architecture [65]

Kandoo [25] is one such representative of the distributed architecture where the authors have used the hierarchical design of the controllers. Figure 2.7 shown above describes the two level of Kandoo architecture. In this architecture, the authors have segregated the data flow into two types. If the data flow is small, it is called small flows, and if the data flow is heavy, it is called elephant flow. Elephant flows are less but can affect the load of the underlying network thus this kind of flow is controlled by the root controller for an optimal path according to the network-wide view. The number of small flows is large, so the controllers at the bottom level will deal with it according to their local view. By doing so, the workload of the root controller is reduced and the adaptiveness of many events increases. The smaller controller can be deployed on the demand of the service and relieves the root controller's burden.

2.3.3 Data Plane

We have discussed several aspects of the application layer and the control layer in previous sections. In this section, we will discuss the attributes of the Data plane. Data plane is also a vital layer where the forwarding of data based on the rules set by the control plane takes place. It is also called the forwarding plane, and its primary function is packet forwarding and supports some types of in-network services to realize various services such as in-network caching and transcoding. However, there are several areas which we understand is important in terms of SDN and are discussed here.

2.3.3.1 OpenFlow Switches

There are several OpenFlow switches available in the industry, and generally OpenFlow v1.0 is used and widely supported. Table 2.2 provides the overview of different OpenFlow switches currently available in the market.

Table 2.2: OpenFlow Switches

	Name	Language	Version
Open Source	Open vSwitch	C/Python	1.0
	ofsoftswitch13	C/C++	1.3
	Indigo	C	1.0
	OpenFaucet	Python	1.0
	Pantou	C	1.0
Commercial	Vendor	Model	Version
	Arista	7050 series	1.0
	Brocade		1.0
	Dell Force10	Dell Force10 Z9000 and SSeries S4810	1.0
	Extreme	Summit X440, X460, X480,and X670	1.0
	HP	3500, 3800, 5400, 6200, 6600, and 8200 series	1.0
	IBM	RackSwitch G8264	1.0
	Juniper	MX-series	1.0
	NEC	NoviFlow 1248, 1132	1.3
	Pica8	P-3290, P-3295, P-3780, and P-3922	1.3
	Larch networks	OpenFlow switch	1.0

2.3.3.2 Data Plane Technologies

- *Packet forwarding*: There are many ways to improve the packet forwarding and switching such as the use of acceleration card to enhance OpenFlow switching. The authors in [39] uses acceleration card for switching and the results show that there is 20% reduction in packet delays while maintaining the comparable packet forwarding throughput compared to other existing designs.

- *In-network services*: In-network services are the applications running at the network application layer or above, and it may consist of the data storage and the manipulation applications. However, as the network connectivity becomes a commodity, the users and applications increasingly demand new in-network services. Data plane, on the other hand, provides programmability to support various in-network services. ClickOS [3] is such example which provides support to numerous applications such as NATs, rate-limiters, application accelerators, etc. It supports by making use of Click modular router.

2.4 Service Composition

In the era of technology where devices are more empowered with resources and capabilities and are utilized in every section of human civilizations. However, with the evolution of the network and networking infrastructure, many devices are connected to each other thus craving a path for the broader collaborative approach to achieve the common goals. In lame words, the interconnectivity of devices can be exploited to produce a collective work without degrading the performance of one device. This concept is referred as service composition. Service composition can be defined as an aggregation of services from different sources to accomplish a particular task or to achieve a business goal. Over

many decades, many types of research have been proposed to achieve the common goal by industry and academic.

Service composition is the process of creating a service by integrating the existing facilities in the network offered by different devices. As devices are more capable of various technologies, there are numerous services which can be combined to achieve a function which can satisfy a business goal. Service composition consists of several phases such as service discovery phase, service integration phase, and service execution. In the distributed environment, the stakeholders are classified into two parts such as participatory nodes or devices and composition initiator node or device. If there is no composition initiator, then it becomes the peer to peer service exchange.

Service discovery and composition has been a major research area for many years. It has been widely studied in terms of web services such as eflow [10], SWEFT [49]. The authors in [22] exploit the collaboration between service brokers and service providers to fulfill inbound cloud service requirement. To address this issue, the authors have proposed agent-based cloud service composition approach, in which the participating cloud and resource are implemented and instantiated by agents. The self-organizing agents use acquaintance networks and contract net protocol to evolve and perform cloud composition.

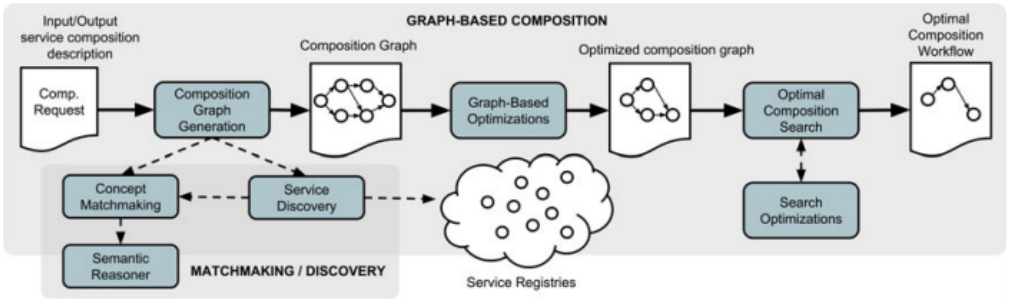

Figure 2.8: An Overview of Graph-based Service Composition Framework [50]

Similarly, the authors in [50] present the theoretical analysis of service composition according to the dependency with service discovery. Furthermore, the authors propose service composition framework with service discovery and enable the graph-based integration of services which are semantically relevant for I/O request. Figure 2.8 depicts the proposed approach, and it also includes the optimal search algorithm to obtain the best composition, and analysis shows the flexibility and scalability of the method and achieves good performance.

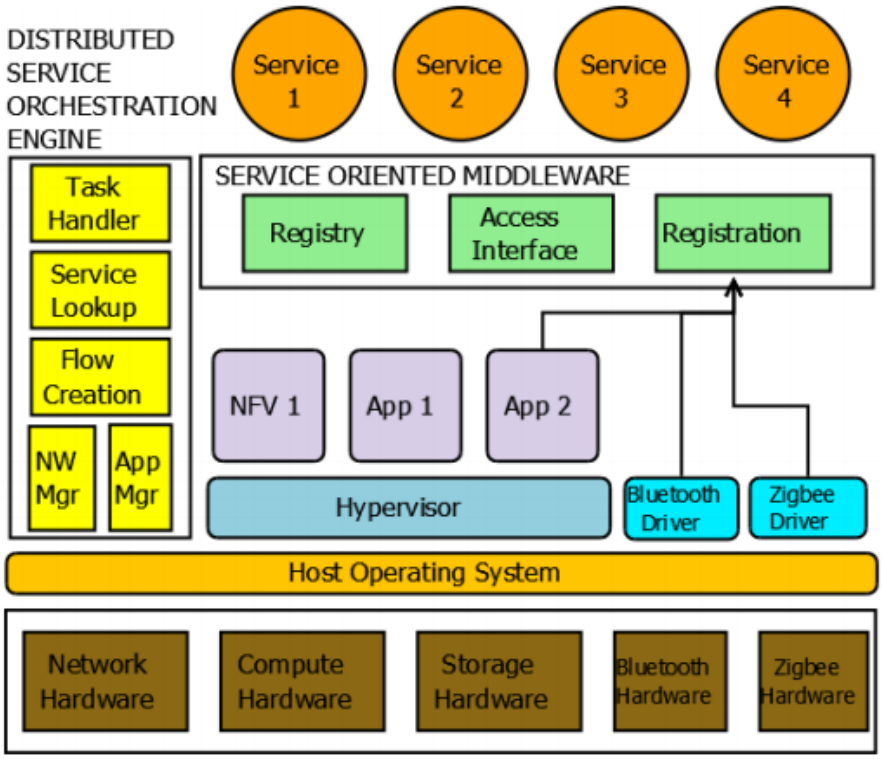

Figure 2.9: An Overview of Software-Defined Fog Computing Architecture [21]

In [21], the authors propose Software-Defined Fog computing architecture which uses Service-Oriented Architecture (SOA). Figure 2.9 shows the middleware and orchestration engine integrated into SDN framework where two significant components are the Service Oriented Middleware (SOM) and Distributed Service Orchestration Engine (DSOE).

SOM abstracts the device functionality and data sources, and exposes them as services on fog nodes which can at the low level provided by the hardware or application module running on the fog nodes. DSOE makes the services discovery by deploying and orchestrating the services and operates to prevent QoS.

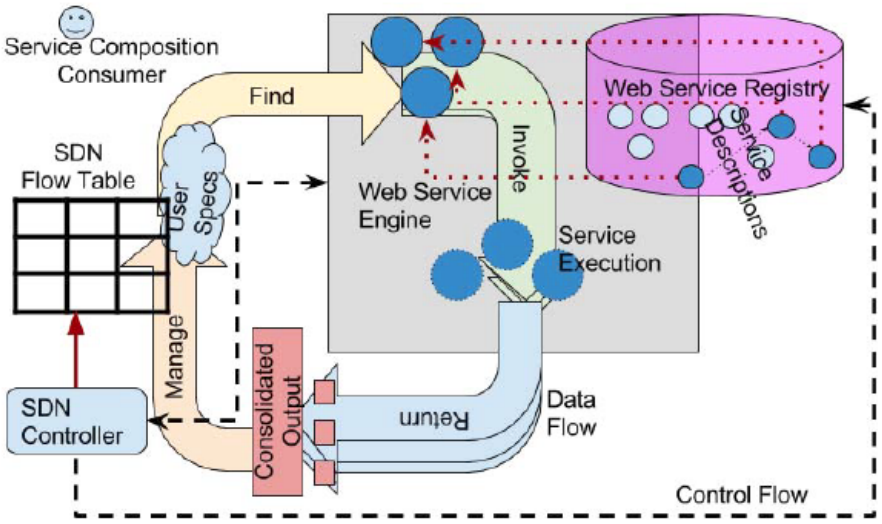

Figure 2.10: A FIRM Procedures [27]

Further, FIRM [27] is another approach for service composition using SDN. It also utilizes the MapReduce [15]. The FIRM procedure comprises Find, Invoke, Return, and Mange as the core procedures. The authors use dynamic programming paradigm. Find procedure finds the appropriate service installations as the core service in the composition. Invoke initiates the efficient deployment of chosen service deployment in the distributed environment. Returns send the results of service composition back to the user. Manage does the management and orchestrates the service composition through a web service registry and SDN controller. Figure 2.10 depicts the FIRM procedures.

Chapter 3

CSP Approach for Computational Offloading

An open problem in achieving a solution for the computational offloading in MEC environment is how to create an automatic and dynamic service composition for computational offloading with limited human intervention. We aim to automate the service discovery for computational offloading and enable the user to decide what services are required. The hope of finding more reactive solution for computational offloading for MEC environment has lead to the emergence of research work specifically targeting SDN-based computational offloading in MEC environment (e.g. [4], [13]). However, such works rely on offloading decision for one service at a time and the effect of which is depletion of device resource (i.e., battery and processing power). Moreover, most of the research work focuses on the bandwidth efficiency or reducing energy consumption but not the user specific requirements.

In this chapter, we present a software-defined service composition approach for MEC environment. We leverage the features of SDN and service composition to solve the offloading problem. We examine the possibility of performing SDN-based service com-

position by modeling and solving the service composition problem as a *CSP*. A CSP is defined by the triplet $(S,\ D,\ C)$, where $S = \{S_1, ..., S_n\}$ is a set of n services, D $=\{D_1, ..., D_p\}$ is a set of p domains of values, and C $=\{C_1, ..., C_k\}$ is a set of k constraints of the problem. A CSP is solved by assigning values to variables whereby all constraints are satisfied. In our extension, we apply into the various phase of service composition, the QoS function taking into account the degree of match between the requirements of a user's task and the qualities and capabilities of service composition, including the current state and availability of each service taking part in the composition.

The resulting contribution of this chapter is a CSP model for task-based and QoS-aware service composition in MEC, utilizing backtracking and forward checking based service selection algorithm for solving the respective CSP. Through simulation and comparison, we show the effectiveness of our method.

3.1 Existing Approach

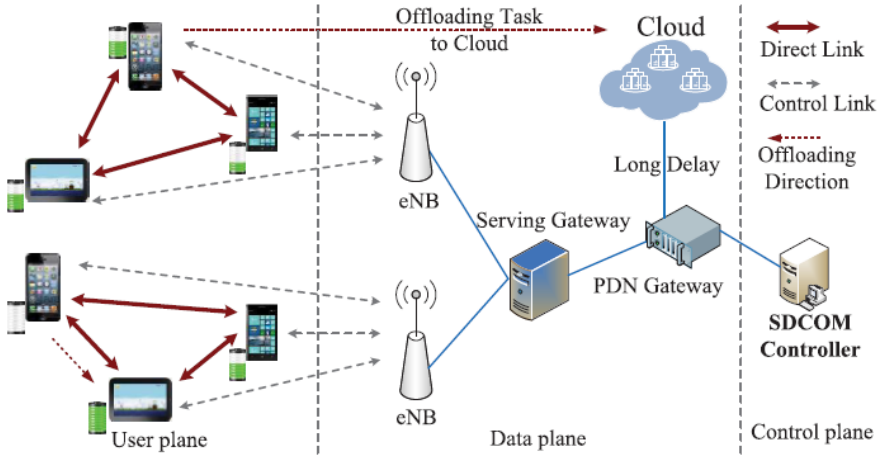

Figure 3.1: Software-Defined Co-operative Offloading Model [13]

The authors in [13] have presented the working model of the Software-Defined Cooperative Offloading Model (SDCOM) which consists of the user plane, the data plane, and the control plane. The SDCOM controller resides on the PDN gateway and provides data management service to the overall network. Figure 3.1 provides the overview of the SDCOM model. The main idea here is to find the optimal solution for offloading the service execution to the cloud or the neighboring nodes. Hence, the offloading can be categorized as two different ways of execution:

- *Cooperative Execution*: Cooperative execution is nothing but the execution of the service by the peer devices and not offloading to the cloud. The offloading decision is made based on some energy consumption model. If the energy consumption of the cloud execution of the service which includes transmission cost and computation cost is higher than the energy consumption of the device execution, then the cooperative execution is selected.

- *Cloud Execution*: In cloud execution, the service execution is done by the cloud, based on some algorithm. Similar to the cooperative execution, if overall energy consumption of service executed by the peer devices is more than the energy consumption of service executed by the cloud, then cloud implementation is selected.

Figure 3.2 shows the offloading procedure where they have n services to execute over an extended period, denoted by set $S = \{S_1, S_2, ..., S_n\}$. According to an energy-saving greedy algorithm, the SDN controller allocates the work to the devices for execution.

The entire offloading procedure is a multi-stage process. A mobile node which has a service for execution, sends a request to the corresponding base station. The base station then forwards the request to the SDN controller. The SDN controller keeps track of all the information about the devices and recently executed services. The controller first

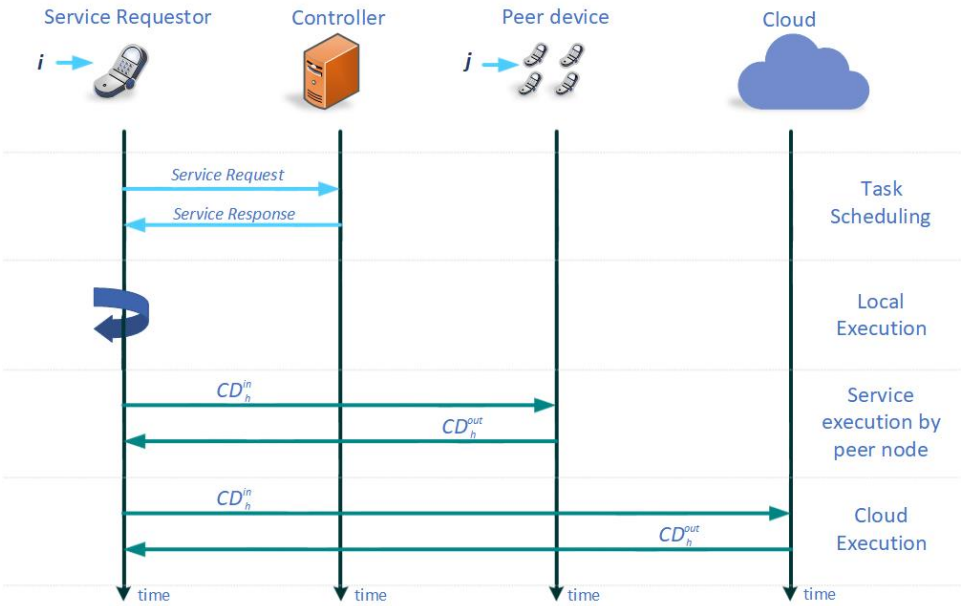

Figure 3.2: Offloading Procedure [13]

checks whether there is a peer device who has recently performed the same service. If yes, then the controller updates the flow table else it executes an energy-saving service-scheduling algorithm. Based on the result obtained from the algorithm, the controller assigns the service to the appropriate service provider.

3.1.1 Pros and Cons of the Existing Approach

The existing approach works toward the efficient cooperation among mobile users for computational offloading. The existing method makes use of the SDN architecture to design the offloading framework which is one of the advantages this approach has. Moreover, this approach works efficiently to save the energy of the devices. However, the existing method does not focus on achieving low latency execution and meeting specific mobile user specifications. Departing from the current strategy, our approach concentrates more on the user-specific execution of the task to achieve low latency.

We further accomplish the collaborative execution of the task in which cloud, service requesting user and peer devices perform parallel execution of the job. The current and proposed approach both follow a distributed computing paradigm where the execution of a task takes place in a distributed manner. Since the current approach has features like obtaining device information and SDN, we take the existing system and modify it to integrate the service composition model. In the present approach, the controller performs the decision of offloading for every task which can be a very exhaustive approach for a large number of data and certainly lag in satisfying specific user requirements. However, in our approach, the controller discovers all the required services at once. We obtain the service composition by applying CSP with QoS model integrated to it. With thorough experiment and analysis, we show that our approach achieves low latency task execution and satisfies user requirements.

3.2 Notations

Table 3.1 shows the list of the frequently used notations in the chapter.

Table 3.1: List of Notations

Notation	Explanation
S	Set of services $\{S_1, S_2, ..., S_n\}$.
N	Set of nodes $\{N_1, N_2, ..., N_m\}$.
D	Set of domain values $\{D_1, D_2, ..., D_p\}$.
R	Set of ranks $\{R_1, R_2, ..., R_k\}$.
T_{Req}	Task requirement list.
C_{Ser}	A set of atomic services.
$S_{Request}$	Service composition request.
CC_{S_i}	CPU cycle required for service S_i.
CD_S^{in}	Input computational data of service S.
CD_S^{out}	Output computational data of service S.
$\Theta_T^{i,j}$	Transmitting power of node N_i to N_j.
$\Theta_R^{i,j}$	Receiving power of node N_i to N_j.
τ_{off}	Offloading time.
τ_{exe}	Execution time.
τ_{N_j}	Total service execution time.
Γ_{N_j}	Computational capability of device N_j.
δ_{N_i}	Up-link data rate of device N_i.
ξ_{exe}	Energy consumption of a device executing a service.
ξ_{N_i}	Energy consumption of node N_i executing a service.
ξ_{off}	Energy consumption of a device offloading a service.
W_{SA}	Weight representing availability of service.
W_{PA}	Weight representing user preference or user-defined weight.
λ_T	Energy consumption during transmission of data.
λ_R	Energy consumption while receiving data.
R	Service rank.
R_{MA}	Mobility-aware service rank.
NQ	Network quality.
MF	Function that maps time points to the user location.
P	Function that maps position to NQ.
SRT	Service response time.
SRQ	Service response quality.
QoS_{MA}	Mobility-aware quality of service.
QoS_{SC}	Quality of service of service composition.
UW_{S_j}	Number of users waiting for a service.
CS_{S_j}	Current state of a service S_j.
G_{S_j}	Guidelines of a service S_j.
utp	User time point.
$N(S)$	Represents N service options of service S.

3.3 Service Composition Overview

The service composition protocol builds on a reactive system and consists of mobile nodes which represent the real-time devices in the mobile environment. Nodes connect to each other via Wi-Fi or LTE network protocol. Each device has different resources like computational power, battery, etc. Any device can act as a service provider, service requester, or as a Composition Manager (CM). Selection of CM depends on the highest degree in terms of the resources, computation capability and the service provider nodes in its vicinity to perform service composition. The requestor node initiates the process by sending the request to CM for composing a service. Upon receiving the request, the CM performs the service discovery in its vicinity. Once the potential service providers are detected, the CM performs the service integration and updates the execution flow for execution of the task on behalf of the requester node.

The process of the service composition is a three stage process. The first stage is the service discovery stage in which CM discovers all the necessary services after receiving the request for service composition. The discovery process is based on matching the requested service with the cached description of the other available services. Each node periodically update CM regarding the services it can offer. CM then matches each cached service information with requested service and selects the best service provider node. However, the discovery process may identify the same service delivered by different nodes or same service of a different type.

The second phase is the service integration phase. Upon discovering all potential services from different nodes, the CM performs integration of the services. There is no specialized algorithm used to select a service, but it depends on the cost such as distance. After selecting the services offered by the nodes, the CM creates an execution

flow identifying node binding, service information, network parameter and other factors related to the computation.

The third phase is the service execution phase, where the execution occurs in a distributed manner. The CM executes one service and transfers information related to the executed service to the next node and vice versa. The CM and service provider nodes perform the overall execution process as per the execution flow.

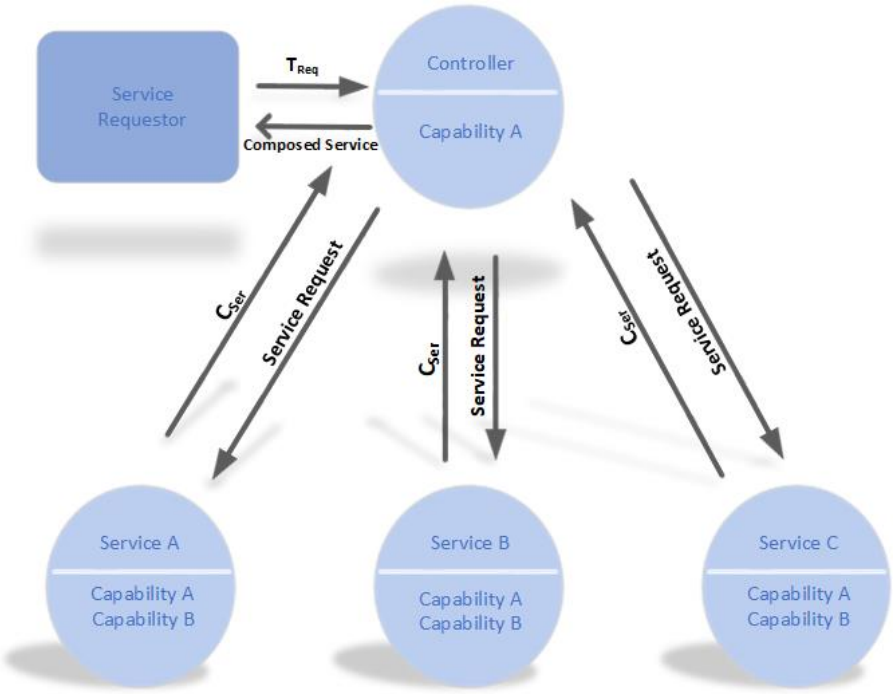

Figure 3.3: Overview of Controller-based Service Composition

We have modified the service composition protocol by inducing SDN controller as CM as shown in Figure 3.3. However, the controller does not offer or execute any service due to resource constraints. To leverage the features of SDN architecture, we propose SDN-based service composition architecture described in the next section.

3.4 Scenario Description

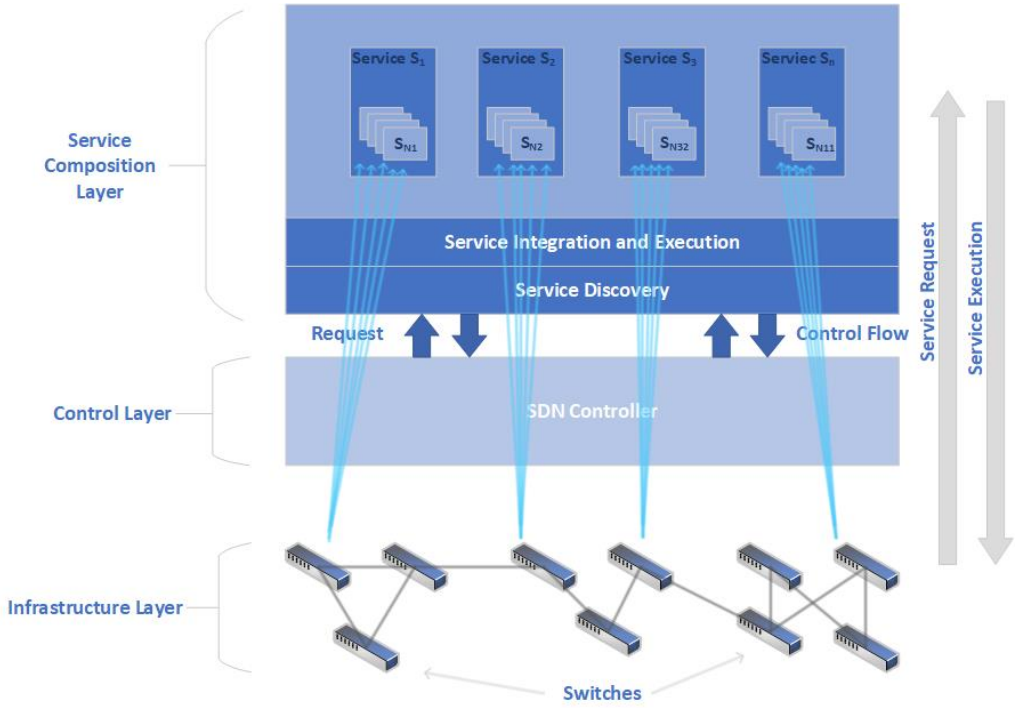

Figure 3.4: System Architecture of Software-Defined Service Composition

In this section, we describe the scenario of SDN-based service composition and discuss the procedure of the proposed architecture. We integrate the SDN framework with service composition and call our architecture as Software-Defined Service Composition (SDSC) architecture. Figure 3.4 provides the high-level architectural design of SDSC architecture. The SDSC architecture and framework is separated from the deployment, offering the loose coupling between logic and implementation. The SDSC defines composition loosely as a compound of multiple executions of services, which can be a computational service and may require parallel execution of services. Furthermore, throughput and efficiency of the job can improve by decomposing them into smaller interdependent services or jobs that can execute independently and later be combined to provide the result. Our

architecture delegates the device registration and management functionalities to the SDN controller extensions, foreseeing a customizable and adaptive service composition for computational offloading. In SDSC, the control plane is completely decoupled from the data plane, as though it is a centralized application rather than a distributed one. However, the execution of the task is distributed among the service provider nodes.

We base our architecture on the following entities:

- Service Requester (SR): A mobile node or device from which the service composition request originates.
- Service Provider (SP): A mobile node or device which contributes its service to the service composition.
- Composition Manager or Controller (CM or C): A node whose sole purpose is it to manage the discovery and integration of the service.
- Atomic Task (Task): An application-level generated task or goal, independent by nature which may requires one or more services to accomplish.
- Atomic Service (Service): A service offered by a service provider node. A service may require multiple components but considered when all the component resides on the same node.
- Task Requirement(T_{Req}): The desired stipulation of service and its properties concerning its execution and execution flow.
- Candidate Service (C_{Ser}): A set of atomic service which when composed satisfies specification mentioned in T_{Req}.

In SDN-based MEC environment, the connection is established between two devices with the help of a controller and data between devices is transmitted directly. As-

sume a mobile user wishes to perform a complex application such as image processing and requires some of the tasks to be executed by the services offered by the devices in it's vicinity, initiates *service request* which includes task requirement T_{Req}. T_{Req} is a specification which explicitly includes task and service requirements. T_{Req} specification represents constraints by which the task should execute like service execution time, number of services required for the task and user preference for any particular service. These constraints act as a means of satisfying both job specification and user's desired QoS. We refer to QoS, such that at any instant, numerous potential compositions exist in the environment satisfying a task. Therefore, separating one composition from another is the QoS associated with the particular composition defined by the user. Hence, we integrate the SDSC architecture with the QoS function.

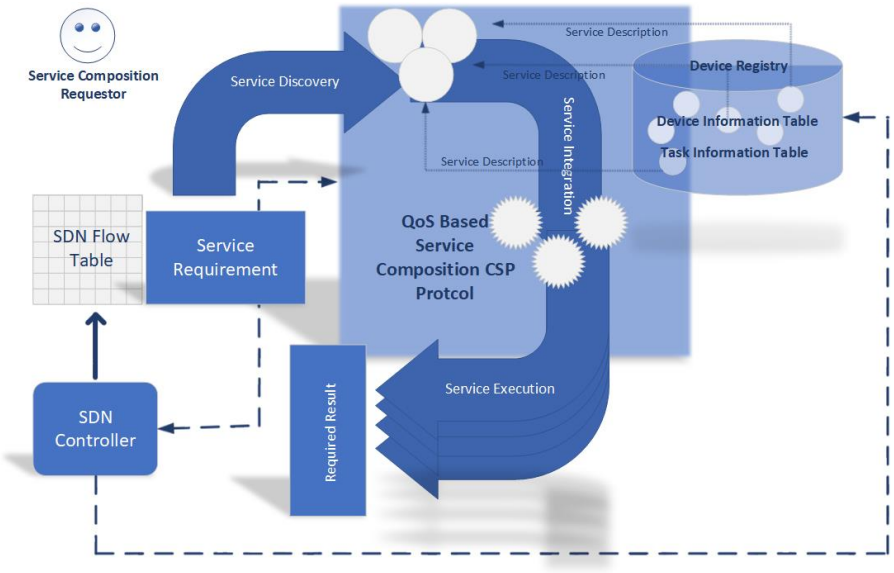

Figure 3.5: Software-Defined Service Composition Procedure

We define the procedures of the SDSC architecture as shown in the Figure 3.5. The procedures involve service discovery, service integration and service execution as the core procedures. However, the device registration process takes place when a new device reg-

isters itself in the SDN network, and the controller records all the information about the device such as services offered. Moreover, device registration is an ongoing process whenever there is a change in the status of the node, and the controller updates its knowledge about it. The information also includes the computation capability, transmission power, residual power, etc., controller stores all this information in *device information table*. The controller also records information about the recently executed task in *task information table*, e.g., which device has executed the task and cached the result. However, due to the limited storage capacity, the controller only stores the fingerprints [58] derived from the recently executed task. CPU frequency, as constant reported only once, but all other device capabilities communicated periodically to the controller. Besides all the variable, the controller must update the *device information table* when a mobile device makes a service request.

Upon receiving the service request from the requester node, the service discovery procedure identifies the services to invoke by analyzing all the services offered by peer nodes from device information table. Service integration phase integrates all the discovered services and provides the best composition possible. Service execution phase executes the service and provides the final output to the requestor node. SDN controller performs the service composition and leverages the network topology and flow table information readily available to it.

Our objective is to maximize the QoS generated by the composed service. It is done by efficiently examining the possible composition in user's vicinity and identifying all the nodes that satisfy the user requirements, adhere all service constraints, and best caterer to the user-defined preferences. The process is complicated by the fact that the nodes are independent and may be operating under different sets of rules and guidelines which is necessary for determining the composition decisions. We refer to such information as

device information and it contains security rules, business model, etc. However, device information is confidential, and devices should not share with each other. In our approach, we use SDN controller to utilize minimal device information and provide efficient service composition without sharing the information of the devices.

3.5 Problem Statement

The goal of our work is to find suitable service provider nodes for the service composition. We aim to find the nodes based on the ranking of the service to maximize the QoS generated by composed service. We model software-defined service composition in the form of a CSP.

We define CSP approach based on the following entities:

- $N = \{N_i\}\,(i \in \{1, ..., m\})$. N is a set of nodes.
- $S = \{S_j\}\,(j \in \{1, ..., n\})$. S is a set of services, where each service S_j is offered by any node.
- $R = \{R_k\}\,(k \leq m)$. R is a set of ranks, where each rank R_k denotes the preference given to the particular node for service S_j. For example, if rank of service S_1 offered by node N_1 is R_1 and N_2 is R_2 $(R_1 > R_2)$ then node N_1 is selected for service S_1.

Further, we model CSP as follows:
$CSP = (S, D, C)$, where

- $S = \{S_i\}\,(i \in \{1, ..., n\})$. S_i is a variable corresponding to the services.
- $D = \{D_i\}\,(i \in \{1, ..., p\})$. D is a set of p domains of values, where each domain D_i provides the option of service provider nodes for any service in S.

- C is the set of constraints of the problem formulated as follows. Let $D_i^S \subseteq D$ be the set of domains corresponding to the service S_i. Let $D_j^S \subseteq D$ be the set of domains corresponding to the service S_j. Then, the following constraints must be satisfied:

$$\forall_{i,j} \quad D_i^S \cap D_j^S = \{\ \} \quad i,j \in \{1,...,n\},\ i \neq j \tag{3.1}$$

$$\forall_i \quad \cup S_i = S \tag{3.2}$$

In other words, Constraint 3.2 implies that each service from the set of service S is assigned to a node. Constraint 3.1 ensures that no two nodes can have the same service. There can be additional constraint depending on the mobile devices taking part.

Having identified our CSP based on service composition, we apply the algorithm based on the backtracking and forward checking [32] to solve the service composition problem in MEC. We call our method as the CSP approach and the respective algorithm used in the protocol as service selection algorithm.

3.5.1 QoS Function

QoS function is the measurement of the overall performance of the service composition regarding ever-changing user environment. For the quantitative measure of the QoS, we consider several user-defined aspects of the service such as availability, priority, and degree of match between user's task requirement and composed service. While defining QoS function for service composition in SDN environment, we consider user's environment and expectation while instantiating a service according to the user's current circumstances.

First, we shall calculate the availability of each potential service taking part in the composition to compute the overall availability of the composed service. We define the availability of composed service as weight W_{SA}, i.e.,

$$W_{SA,S_j} = \det(UW_{S_j}, CS_{S_j}, G_{S_j}, ...), \quad where \quad S_j \in S \tag{3.3}$$

$$W_{SA} = min\Big\{W_{SA,S_j}\Big\} \tag{3.4}$$

In Equation 3.4, we calculate the availability W_{SA} of service composition by estimating the availability W_{SA,S_j} of each service contributing to service composition using determinant of the matrix and taking the minimum value. The estimation of availability of each service is based on various factors related to the service such as the number of users waiting for the service UW_{S_j}, the current state of the service CS_{S_j}, the guideline of the service G_{S_j} and other availability related information of the service. We further consider W_{PA} as a weight given by the user to specific service. Now we calculate the overall QoS of service composition QoS_{SC} as,

$$QoS_{SC} = max\Big\{W_{SA} \cdot \sum_{j} W_{PS_j} \cdot Sim(S_{j,sc}, S_{j,T_{Req}})\Big\}, \tag{3.5}$$

where similarity function Sim is used to measure the similarity between the service S_j in composition and T_{Req}. However, the output of similarity function is between zero and one which is a normalized output considering the negative and positive both variations from an exact match, where the closer the value to the one, the higher the degree of similarity. The calculation of the similarity measure iterates for all the services and each applied with the user-defined weight W_{PS} given to specific service. We then perform summation and apply the service availability weight W_{SA} calculated by Equation 3.4. Maximum among all the values is our QoS_{SC}.

3.5.2 Energy Consumption while Computation of Service

During the task execution, CPU dominates the energy consumption of the mobile devices. CPU energy depends on the CPU type, clock frequency, workload, etc. We make use of the Miettinen et al.[44] depiction of the CPU energy consumption from the perspective of computing efficiency. The number of CPU cycle CC_{S_i} need to execute the service S_i, depends on the input data size and type of the task. We formulate the CC_{S_i} as,

$$CC_{S_i} = \int_T (CD_S^{in}), \quad where \quad S_i \in S \tag{3.6}$$

where the function $\int_T(\cdot)$ determines by the service type T. The CPU cycle depends on input data size and type of service. CD_S^{in} is the computation data input of service S_i. We formulate the energy consumption of device N_i for executing a service S_i as,

$$\xi_{exe} = \int_{N_i} (CC_{S_i}), \quad where \quad S_i \in S \tag{3.7}$$

where function $\int_{N_i}(\cdot)$ is the power coefficient of N_i 's CPU usage. If device N_i has already executed the service S_i of the same type and cached the results, then the energy consumption of device N_i for service S_i is zero.

3.5.3 Energy Consumption while Offloading the Service

In wireless network, energy consumption of offloading a service plays a vital role in determining the preference given to the node for selection in composition. We adapt the work of Kumar et al.[31] for power consumption model from node a to b which we feel very well suited for the MEC model. We formally represent the power consumption of offload a service as,

$$\lambda_T = \Theta_T^{i,j} \cdot \frac{CD_S^{in}}{\delta_i} \tag{3.8}$$

$$\lambda_R = \Theta_R^{i,j} \cdot \frac{CD_S^{out}}{\delta_j} \tag{3.9}$$

where $\Theta_T^{i,j}$, $\Theta_R^{i,j}$ are transmitting and receiving power of node i to j respectively. δ_i and δ_j are transmission rate of the link N_i and N_j respectively. CD_S^{in} is the incoming computation data and CD_S^{out} is outgoing computation data.

In the process of calculating energy consumption of potential service composition model, we call service requester as node N_i and service provider as node N_j.

Definition 1 *Let node N_i be the service requester node and node N_j be the service provider node. Therefore, the offloading energy consumption of the service requester node is sum of the energy required to transmit the computation data from node N_i to node N_j and receiving the data from node N_j to node N_i, i.e.,*

$$\xi_{off} = \Theta_T^{i,j} \cdot \frac{CD_S^{in}}{\delta_i} + \Theta_R^{i,j} \cdot \frac{CD_S^{out}}{\delta_j} \tag{3.10}$$

Definition 2 *The total energy consumption of a node is the sum of energy consumed in transmission as well as energy consumed in computation, and transmission includes receiving the data and sending the output, i.e.,*

$$\xi_{N_i} = \Theta_R^{j,i} \cdot \frac{CD_S^{in}}{\delta_i} + \xi_S^{N_j}(S_i^{d_i}) + \Theta_T^{j,i} \cdot \frac{CD_S^{out}}{\delta_j} \tag{3.11}$$

3.5.4 Time Consumption while Offloading and Execution of Service

In MEC environment, the latency is a significant factor. We introduce the latency as a constraint to satisfy the QoS. We take the work of Chen et al.[11] as an example, who proposed a computation model which we feel is well suited to the service composition model. However, unlike Chen et al. we make use of the overall time consumption in the execution of task as one of the constraints to detect appropriate candidate for service composition.

We formulate the execution time of a service by a node N_i as,

$$\tau_{exe} = \frac{CD_{N_j}^{CC}}{\Gamma_{N_j}} \tag{3.12}$$

where $\tau_{N_j}^{C}$ is the execution time on N_j requested by N_i. $CD_{N_j}^{CC}$ is the CPU cycle required by computational data on N_j. Γ_{d_j} is the computation capability (CPU cycles per second) of N_j. Equation 3.12 does not include time consumed while offloading task from one node to another. Let us take δ_i as the up-link data rate of mobile device N_i and δ_j as the up-link data rate for N_j. $CD_{N_j}^{in}$ as input data size for device N_j.

We formulate the time required in offloading the task as,

$$\tau_{off} = \frac{CD_{N_j}^{in}}{\delta_j} \tag{3.13}$$

Definition 3 *The total time required for the execution of a service is the sum of the data transfer time and computation time, data transfer time include sending data to service provider and receiving output data, i.e.,*

$$\tau_{N_j} = \frac{CD_{N_j}^{CC}}{\Gamma_{N_j}} + \frac{CD_{N_j}^{in}}{\delta_j} + \frac{CD_{N_j}^{out}}{\delta_j} \tag{3.14}$$

3.6 CSP Approach

The proposed service composition approach allows the controller to form the best possible composition of service at the time of need with regards to the user's service-based desires. The controller acts as a composition manager in deciding the participation of each node and provides a general solution for incorporating device information.

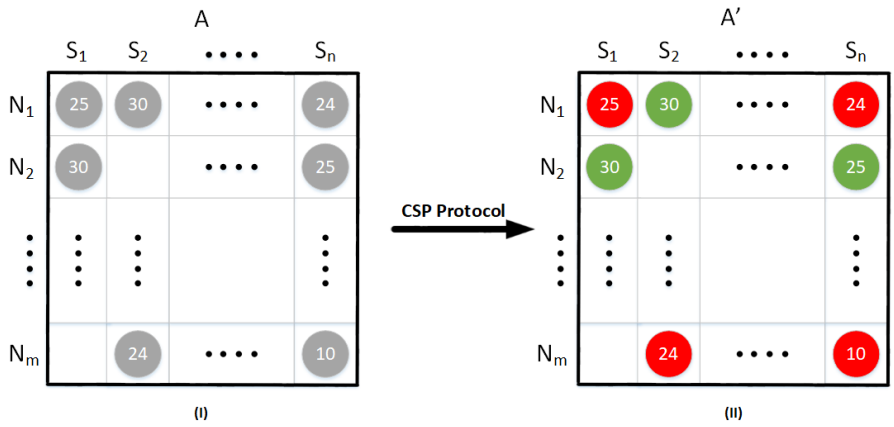

Figure 3.6: Transformation from Matrix A to Matrix A'. Matrix A' shows the abstraction of service composition.

The service composition problem, from a CSP perspective, can be represented as a $m \times n$ matrix (Figure 3.6). This is similar to solving $m \times n$ sudoku [54] problem. However, in service composition problem the rows comprise of m nodes and services as n column, such that:

- A grey circle in Matrix A represents service S_j offered by node N_i. Each service is provided with rank denoting the preference of the nodes. For example, service S_1 offered by node N_2 has rank 30. The solution to the service composition problem requires the application of the constraint set C defined in Section 3.5, whereby a Matrix A' (Figure 3.6(II)) is formed such that:

- A green circle in Matrix A' containing the rank, represents a contributing service S_j for service composition offered by node N_i, such that N_i has committed to contributing service S_j in the service composition.
- A red circle in A' containing the rank, represents a potential service S_j offered by node N_i, such that N_i has the potential to contribute service S_j in the service composition. However, due to its low ranking it is not selected for composition.
- Each column must contain exactly one green circle representing committed node.

The composition of node/service pairs corresponding to the green circle containing the highest rank forms a composition satisfying the constraint set C. The CSP protocol establishes a Matrix A and provides a concurrent and asynchronous negotiation mechanism to transform Matrix A into Matrix A'. The following presents the various steps of CSP approach.

3.6.1 Service Discovery

3.6.1.1 Node Selection

The formation of the node table is the first process of the approach, and it allows the creation of the Matrix A (Figure 3.6.(I)). When there is a requirement for the task execution by a node, the respective node sends service composition request ($S_{Request}$) to the respective SDN controller for the possible service composition. $S_{Request}$ contains T_{Req} which is used by the SDN controller as a specification list to compose the service and decide whether an arrangement is feasible or not. SDN controller scans the stored device information table and task information table. Once SDN controller has investigated for

the potential candidate for the service composition, it creates the node table containing device addresses and a unique identifier of all the possible nodes. The node table is organized in chronological order.

3.6.1.2 Service Ranking

The SDN controller generates the node table which implies the participation of possible services in the service composition by the nodes. For each service S_j, a node N_i has potential to contribute. A service selection ranking is developed using the principle of Equation 3.5 as R, where a similarity function is computed between S_j and the equivalent service requirements specified in T_{Req}. Applied to the similarity function are the availability of service W_{SA} and user-specific/default weights W_{PS} given to the service S_j. Rank of service S_j offered by N_i can be formulated as,

$$R = W_{SA,S_j} \cdot W_{PS_j} \cdot Sim(S_j, S_{j,T_{Req}}), \tag{3.15}$$

At least one device potentially contributes towards a service. However, if any device does not offer a service, then it is offloaded to the cloud. We also consider cloud as one of the participating nodes. For each service, we show service ranking and colors reflecting the contributing service and non-contributing service as shown in Figure 3.6(II). At each $A_{i,j}$ denoting a service, a service ranking is stored, such that $A_{i,j} = (R)$.

Figure 3.7.(I) represents a global view of the Matrix A, following the node table formation in the previous section and included are the respective service selection ranking. Looking at node N_1 and its services, $A_{1,1}$ = (R2), $A_{1,2}$ = (R1), $A_{1,3}$ = (R2).

3.6.1.3 Service Selection Algorithm

We apply service selection algorithm, which is based on the backtracking and forward correction algorithm to solve CSP on the generated Matrix A (Figure 3.6(I)). The SDN controller performs the execution of the algorithm on Matrix A.

In our approach, we refer nodes as service options. Service selection algorithm takes input as a set of service S and service options N for each service. It stores all the service options for each service represented as $N(S_i)$ in $N^{'}(S_i)$ to preserve the original values of service option. The service selection algorithm has two phases: - First, it selects the highest rank node for service S_i from $N^{'}(S_i)$ as mentioned in Algorithm 2. In this algorithm, we first check the consistency of selected node with previous assignment; if consistency found it investigate all the uninstantiated services and its service options and prune all its value that has inconsistency with current assignment. If consistency not seen, it moves to the next highest ranking node. If there is no service option left we return *null* else return the service option for the service.

Second, Algorithm 2 returns the value to the respective service in Algorithm 1. If Algorithm 2 returns null for the service S_j which means that the Algorithm 2 does not find the consistency between the previously assigned node and any future nodes then Algorithm 1 performs backtracking. If Algorithm 2 returns the service option, Algorithm 1 moves to the next service. For every inconsistency in service options of the services, the Algorithm 2 performs pruning of the disparity service options of all variables. Algorithm 1 reaches inconsistency when all services and its service options are exhausted. The exhaustion leads to the termination of the algorithm.

A solution presents itself when all services are consistent states according to the constraints. The controller consistently performs the algorithm till it assigns all the services with service options or it reaches inconsistency.

Algorithm 1 Service Selection Algorithm

Input: Set of all services: S;
Set of service options for each service: N
Output: Set of instantiated value of services: A;

1: Store all service option $N^{'}(S_i) \leftarrow N(S_i)\ for 1 \leq i \leq n$;
2: Initialize service counter $i \leftarrow 1$;
3: Initialize assignment $A = \{\quad\}$;
4: **while** $1 \leq i \leq n$ **do**
5: Select a service S_i from S;
6: Instantiate $S_i \longleftarrow SelectNodeValue()$;
7: **if** S_i is null **then**
8: Remove S_i to A;
9: $i \leftarrow i - 1$;
10: **else**
11: Add S_i to A;
12: $i \leftarrow i + 1$;
13: **end if**
14: **end while**
15: **if** $i = 0$ **then**
return "Inconsistency";
16: **else**
return A;
17: **end if**
18: end procedure

Algorithm 2 Procedure $SelectNodeValue()$

Output: A value in $N^{'}(S_i)$ consistent with A; or null,if none;

```
1: while N'(S_i) is non Empty do
2:     Select highest ranked node a ∈ N'(S_i) and remove a from N'(S_i);
3:     if a is consistent with A according to the constraints then
4:         for all k, i < k ≤ n do
5:             for all values b in N'(S_k) do
6:                 if b is not consistent with a then
7:                     Remove b from N'(S_k);
8:                 end if
9:             end for
10:            if N'(S_k) is Empty then
11:                Reset each N'(S_k) to its value before a was selected;
12:            else
           return a;
13:            end if
14:        end for
15:    end if
16: end while
           return Null;
17: end procedure
```

3.6.1.4 Sample Execution

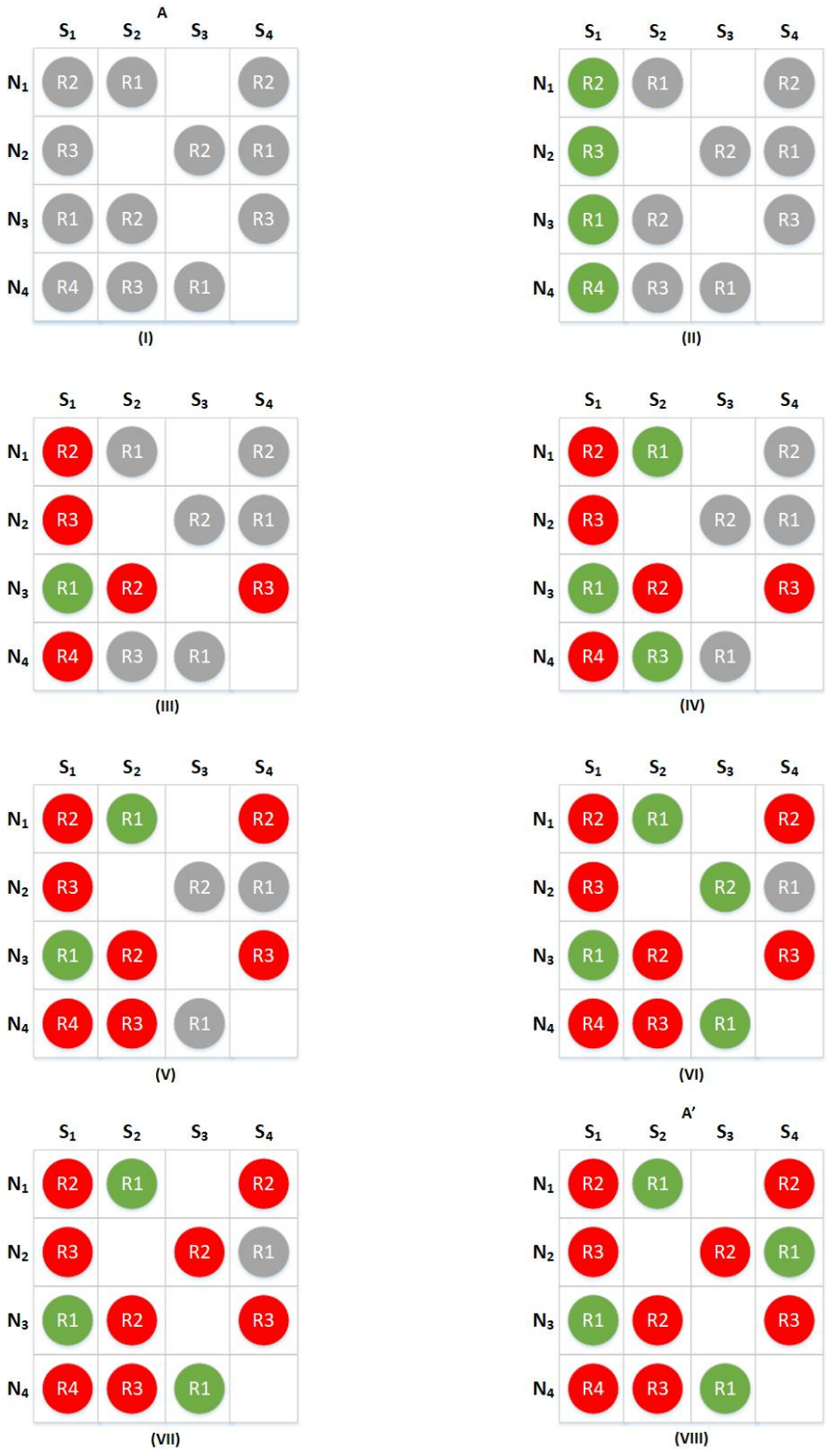

Figure 3.7: Sample Execution of Service Selection Algorithm

Let us consider the Matrix A from Figure 3.7(I), it is a 4×4 matrix which has nodes N as rows and services S as columns. Each node N has some service to commit for service composition. Service selection ranking R is set for each service S which determines the priority of service offered by a node where $R1 > R2 > R3 > R4$. Now centralized algorithm is executed by the controller to obtain an optimal solution for composition.

As S_1 is first in the service requirement list, its service options are considered first where the rank of N_1 is $R2$ for S_1, N_2 has rank $R3$, N_3 and N_4 has rank $R1$ and $R4$ respectively (Figure 3.7(II)). However, N_3 has highest rank, so controller considers N_3 for service S_1. Upon selecting N_3, controller investigates future services and their options, and removes service options which have a discrepancy with the selection of N_3. Therefore, N_3 is eliminated from the options of S_2 and S_4 (Figure 3.7(III)).

Moving forward to the service S_2, which is offered by N_1, N_3, and N_4 (Figure 3.7(IV)). However, N_3 is already eliminated from the options of S_2 because it has committed for S_1. Therefore, N_1 is selected as it has higher rank then N_4 (Figure 3.7(V)). Since N_1 has committed to S_2, controller eliminates it from the options of S_4. Similarly, for S_3, N_4 is selected because it has higher rank then N_2 (Figure 3.7(VI)).

S_4 is last in the service requirement list which can be offered by N_1, N_2, and N_3 (Figure 3.7(VII)). However, due to the forward correction procedure of our algorithm, N_1 and N_3 have already been eliminated from the options of S_4. Therefore, N_2 is selected for S_4 (Figure 3.7(VIII)).

In clarifying the proposed algorithm, we have provided the trace of the algorithms 1 and 2.

3.6.2 Service Integration and Execution

At this point, we have already found the possible nodes and their services which may satisfy user's need by composing the service. However, all the devices are not the best fit for the service composition. The reason behind is the connectivity and bandwidth issues related to the devices which are high in wireless networks. Therefore, the integration phase deals with this issue by building an overlay network which performs the necessary processing.

The service composition requires binding of the nodes for the execution of the task. The controller updates the flow table according to the execution flow provided by the requestor node. The computation of job occurs in a distributed manner, where each participating node carries out its execution and forwards the result to the requestor node in the specific order.

3.7 Evaluation and Analysis

Table 3.2: Simulation Parameter Summary

Duration	30 service requests
Number of controllers	1
Number of nodes	5,10,15,20,25,30,35,40,50
Number of hop	1
Servicer composition length	3 and 5
Computation power of nodes	0.5 GHz, 0.8 GHz and 1.0 GHz
Computation Data	5000 KB
CPU cycle required for Computation	1000 Megacycles

Integration of service composition protocol into a MEC environment using SDN has been simulated, and the result shows the efficiency of the task offloading model. The SDN controller and infrastructure devices or nodes represent the core of the network. The primary goal of this simulation is to study the effect of CSP to find the optimal nodes for the task offloading in SDN-based service composition environment. We performed simulations using MATLAB and Java Software Development Kit (Java SDK). MATLAB provides the variety of libraries which were helpful to create the desired topology for the network events.

The evaluation and analysis consists of four sections. Section 3.7.1 discusses the simulation and schematic of the SDN network that is simulated. Section 3.7.2 mainly focus on the methodology and algorithm of the simulations, and Section 3.7.3 presents and discusses the results. The last section is the conclusion.

3.7.1 Simulation Schematic

The selected network is close to the real SDN network topology consists of the devices, switches and SDN controller. The simulation covers the device node and the controller as the workspace node. The workspace node is the programming part of the simulation that functions as the controller, receives the request from requester device, gets the device and service profile of the other connected neighboring devices, calculates the optimal nodes for the service delivery and updates the routing table.

Table 3.3: Transmission Rate and Power of Mobile Device

Downlink(Mbps)	P_r(W)	Uplink(Mbps)	P_t(W)
20	2.327	4	3.041
16	2.119	3.5	2.822
12	1.911	3	2.603
9	1.756	2	2.16

Table 3.3 shows the transmission rate and power consumption of mobile device over a network. Each node has been specified randomly with transmission rate and power consumption from this data set. We assume that all the nodes are connected and registered to the SDN controller. For the computation of the task, we consider the face recognition application in [56], where the data size of the work is 5000 KB and the total number of CPU cycle required is 1000 Megacycles. Table 3.2 provides the summary of simulation parameters used in our experiment.

3.7.2 Simulation Methodology

We built the simulation in sections, and the combination of these sections represents the whole network. The first section was to create the network by using MATLAB was discussed in Section 3.7.1. Then each node is set with CPU cycle and power. The service requestor node sends the request service event to the controller with the specification list which contains the required service information such as computation cycle, and order of the service on which all the services should execute. At the core of this simulation is the controller function that operates the network. The primary function works as the controller and its job is to the find the optimal nodes in the network which can perform the task based on the requirement specified by the requestor node. Upon selecting the optimal node, the controller informs the requestor node about the participating node which can offer the services and updates the flow table to process the service.

The node generates the requirement specification list and stores it in matrices. Requirements mentioned are pre-segregated based on the subtasks. Face recognition application is divided into specific services which can be offered by peer nodes or the cloud. For this simulation, we have taken it as five services. Each service is mentioned in simulation as the required CPU cycle to execute the service. Each participating node is initialized

with specific service which they can offer and are specified as CPU cycle. Upon registration of a device or node in the network, all the information about the device profile, service provided and the pre-executed task is stored in a list by the controller, and it is randomly updated or when there is a service request. When a node makes a service request, the controller function updates the device list. We assume that controller has considerable amount of storage capacity to store this information.

The controller function is built separately on Java libraries. When a request event arrives, the controller function takes care of the event. The controller performs the backtracking based service selection algorithm as mentioned in Section 3.6.1.3 to select the appropriate node. Before this, each service of each node is given a rank based on the similarity function discussed in Section 3.6.1.2. No two service nodes should have same similarity values for the particular service. If two nodes have the same similarity values that the ranking will be set based on the arrival time. Hence the matrix will not have the same ranking for the same service offered by two nodes. Priority will also be given to the node which has already performed the requested service within a specified period.

3.7.3 Results and Discussion

These results represent the task execution in the simulation environment. We compare our result with the state of the art result of Energy-Saving Greedy Algorithm (EGA) and Conditioned Optimal Algorithm (COA) described by Yong et al. [13]. We used EGA and COA algorithm for service selection in service composition model and used as a benchmark solution. We have also used a random scheduling algorithm as a performance benchmark, where the task offloading decision is a random permutation of the candidate selection.

3.7.3.1 Effect of Data Size on Response time

Figure 3.8 depicts the relation between the various data size and response time. It shows that the response time increases with increase in offloading Data Size. The COA and EGA are focused on energy saving instead of the total response time of a task. The results show that there is a significant decrease in response time when compared to cloud execution, random task offloading algorithm, COA, and EGA. This is because the CSP approach selects the nodes based on the requirement and the selected nodes are the best nodes available in the network with optimal response time. However, in the random selection process, the arbitrary nodes are selected. Cloud execution always has the higher response time since there is an extra time involved in offloading and downloading the data as compared to the peer node. On the other hand, EGA and COA selects the optimal energy nodes.

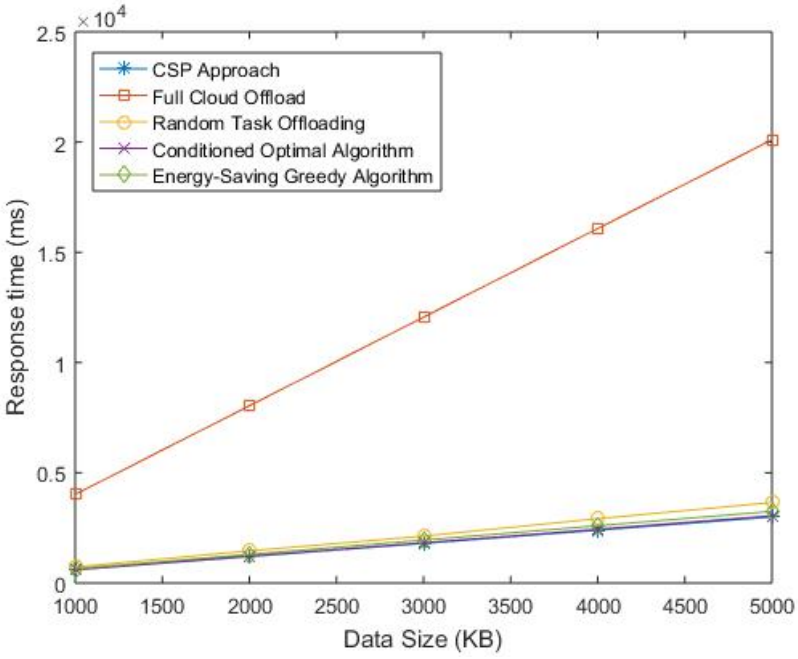

Figure 3.8: Response Time vs. Data Size

3.7.3.2 Impact of Network Load on Response time with respect to Data Size

In the last experiment, we analyze the effect of data size on response time without applying any network load. Figures 3.9 and 3.10 show the effect of data size on response time at 50% and 90% network load respectively. The CSP approach outperforms EGA, COA and random task offloading approaches. The network load has the significant impact on the upload transmission speed since it causes offloading delay. The offloading delay is standard for all the methods.

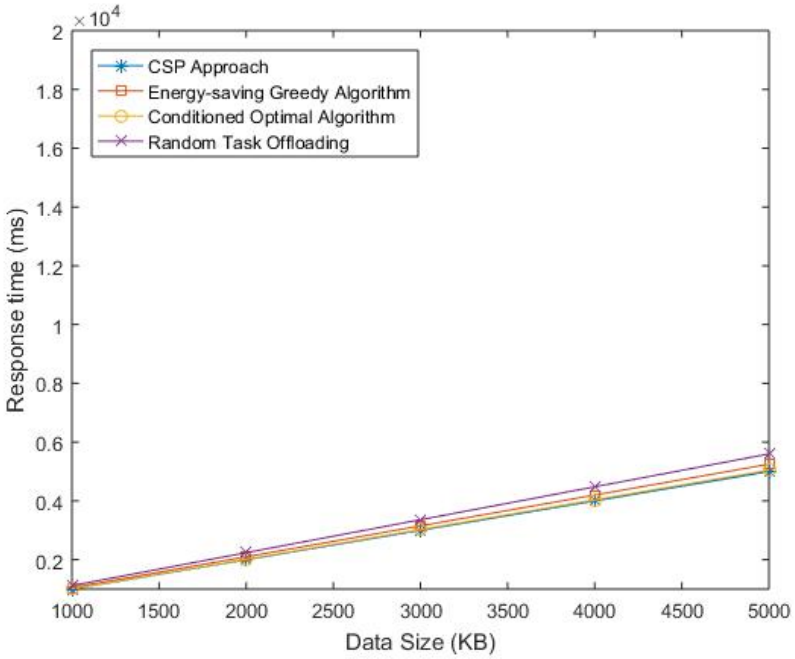

Figure 3.9: Response Time vs. Data Size with 50% Network Load

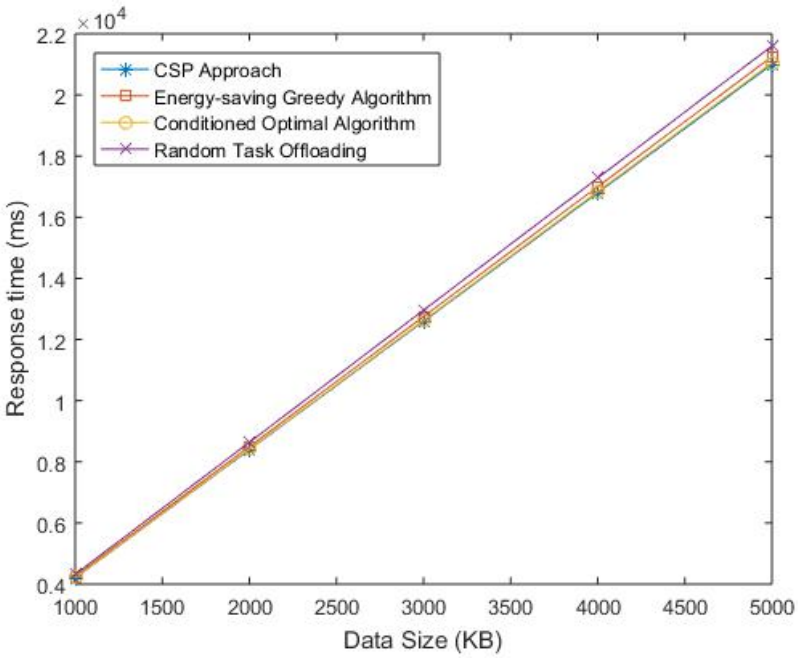

Figure 3.10: Response Time vs. Data Size with 90% Network Load

3.7.3.3 Effect of Data Size on Energy Consumption

In Figure 3.11, we depict the energy consumption concerning different offloading data size. Both COA and EGA achieve better performance than CSP approach and Random task offloading as both COA and EGA algorithms select the most energy efficient nodes. However, the difference between CSP and other two algorithms is less than 10% because the CSP approach works with finding the optimal node for service composition keeping both response time and energy consumption as a requirement.

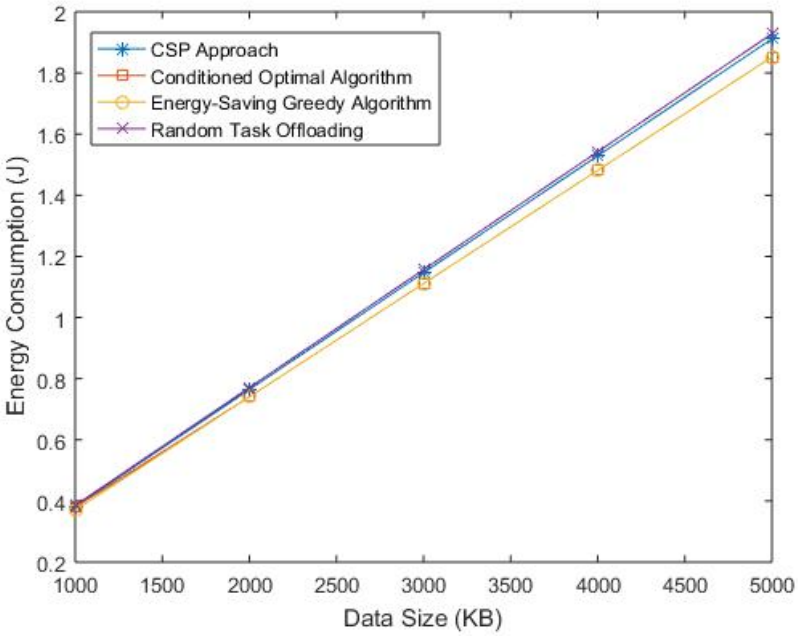

Figure 3.11: Energy Consumption vs. Data Size

3.7.3.4 Effect of Number of Nodes on per Node Execution time

Figure 3.12 shows the per-node execution time concerning increase in the number of nodes. The CSP approach shows that optimal node selected based on the constraints have the significant amount of the decrease in per node execution time, whereas the random approach shows that the decline in per node execution time but has higher values than the CSP approach, which is due to the fact that the selection of node is random.

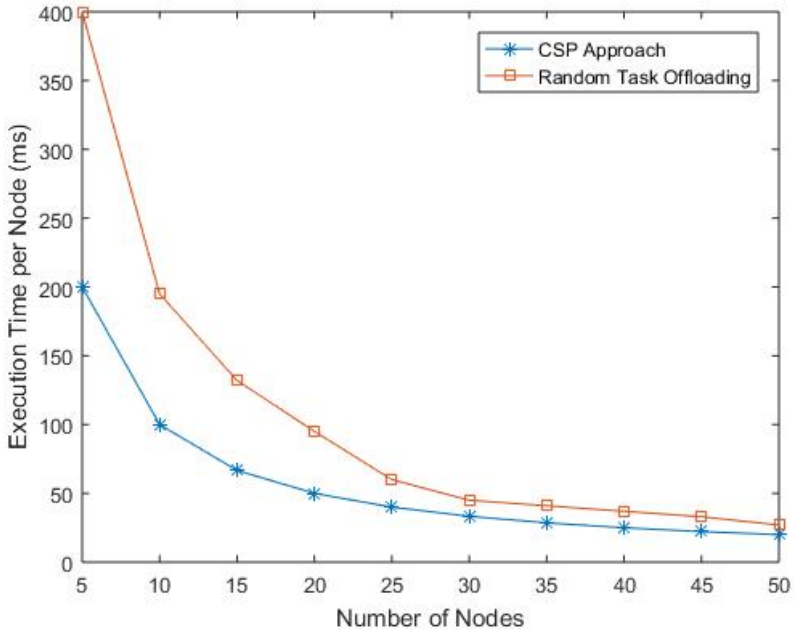

Figure 3.12: Execution Time per Node vs. Number of Nodes

3.7.3.5 Effect of Number of Nodes on Cloud Execution Time

The CSP approach has the significant impact on the cloud execution as the algorithm selects the optimal nodes and cloud-based execution on the user requirement. We have performed the simulation taking cloud as a node with the reasonably high processing power of 10 GHz. We added data transfer rate for the cloud as Upload and Download speed, which is 2 Mbps and 9 Mbps respectively. The task is offloaded by the node when the execution is not feasible. Figure 3.13 shows the effect of CSP approach on cloud execution concerning increase in the number of nodes. The cloud overhead significantly decreases with both Random approach and CSP approach. However, the CSP approach has more impact on cloud overhead as the offloading decision is based on QoS ranking of the services offered by nodes including cloud.

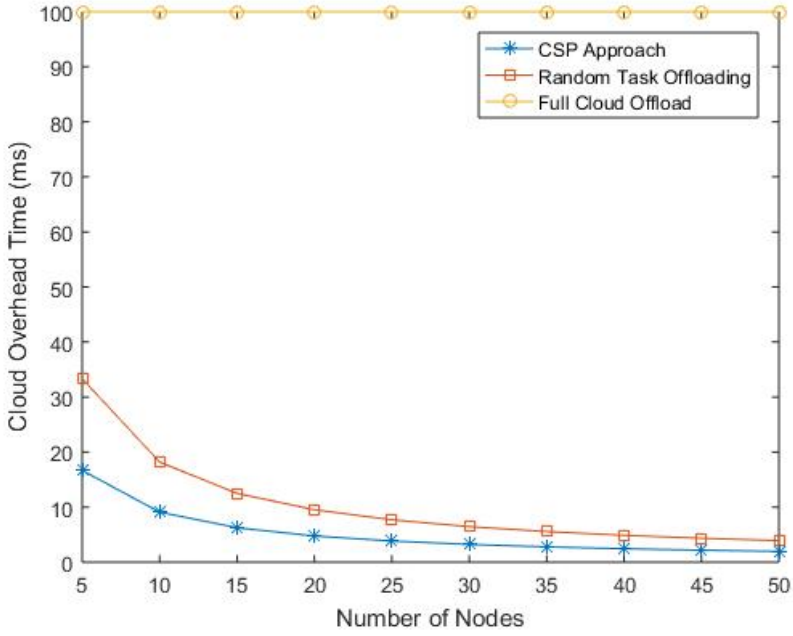

Figure 3.13: Cloud Overhead Time vs. Number of Nodes

3.7.3.6 Effect of Data Size and Number of Nodes on the Percentage of Requirement Match

Figures 3.14 and 3.15 show the effectiveness of CSP approach in meeting the requirement by the user. The matrix used in measuring QoS represents as requirement matching percentage. As mentioned in the Section 3.5.1, the degree of match between the composed QoS and the best possible QoS currently present in the network. The results indicate that the performance of random approach degrades with the increase in data size and the number of nodes whereas the CSP approach results in more than 98% in matching the constraint constantly. This is because CSP approach always find the nodes closest to the user requirements based on Equation 3.15. Figure 3.14 shows the percentage of constraints matched for different data sizes. Figure 3.15 shows the percentage of constraint matched for different number of nodes.

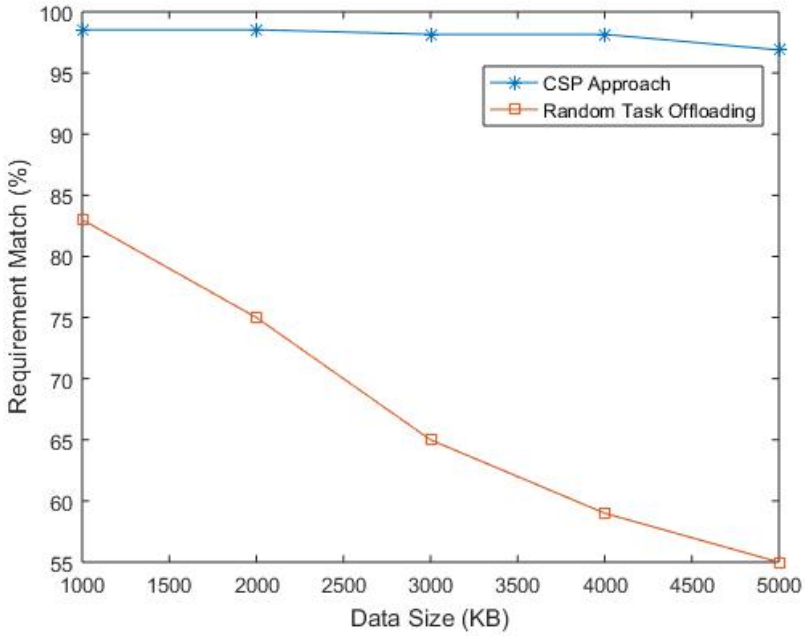

Figure 3.14: Percentage of Requirement Matched vs. Data Size

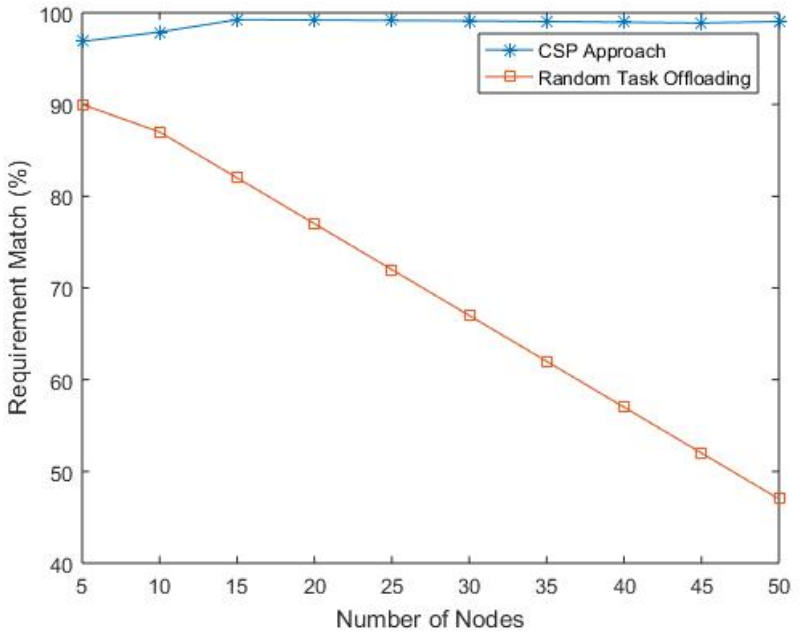

Figure 3.15: Percentage of Requirement Matched vs. Number of Nodes

3.8 Conclusion

We leverage the SDN to design our software-defined service composition framework for finding the solution for efficient offloading approach among mobile users, which aims to provide the user-centric composition of service and reduces latency. Based on the device information such as computational capability, a controller decides the distribution of task periodically. We formulate the user-centric service composition as a CSP and design the algorithm which selects the optimal devices for service offloading and execution. A thorough evaluation of our approach shows that the CSP approach not only satisfies the user requirement but also does the execution of user task with a low response time. Through evaluation, we also revealed that the CSP approach reduces the execution time of services, reduces the burden on the cloud service, and meets the user-specific requirements as compared to the random, conditioned and greedy service selection algorithms.

Chapter 4

Mobility-aware Service Composition

Due to the increase in demands of various IoT-based environments such as Fog computing and MEC, time consumption in performing the task either by cloud or the edge devices is critical for an acceptable delay in 5G and LTE. However, the problem with maintaining the low latency service execution is how to tackle the user's mobility. Therefore, an appropriate mobility prediction method is needed to estimate the system state change in future, to let the system adapt and minimize the delay. In our case, to provide seamless service composition, mobility prediction information could easily be used to decide the selection of the service provider nodes or devices. Prediction of mobile user's location is essential for the wide range of mobile applications. In the cloud-based infrastructure, different virtual services can use this prediction to optimize the network performance. Radio Access Network (RAN) can also exploit the prediction information to deploy, upgrade or manage various services and resources. In overall, seamless service and service continuity can be used to support with multiple optimized traffic forwarding mechanism that considers the location of users.

In the previous chapter, we addressed a dynamic service composition problem by developing SDN-based service composition architecture. Moreover, we utilize CSP ap-

proach for task-based, QoS-aware service composition. We use backtracking and forward checking based service selection algorithm to solve the respective CSP. The benefit of using CSP approach is the ability to provide composition by satisfying all the constraints and selecting the best fit service provider by measuring the similarity between required service and potential service in the composition.

However, lacking from the composition model is the awareness of user's mobility. In this chapter, we propose a mobility-aware approach for service composition where controller makes use of the prediction method to predict user's future location and network quality at that position. We further specify a mobility-aware QoS computation rule that allows the computation of QoS as a degree of match between user's service requirement and service in composition, and response quality of the service. We further make use of the mobility prediction method to find all the potential service provider nodes and develop a ranking for each service provider node for all the required services.

The resulting contribution of this chapter is to develop location-based node selection, mobility-aware service ranking, and QoS model. However, we follow rest of the approach and algorithm as described in Chapter 3. Through simulation and comparison, we show the effectiveness of the mobility-aware approach.

4.1 Mobility-aware Approach

Mobility prediction is essential for the service composition in MEC as the requestor device is not static all the time. Therefore, the composition of service made by keeping the requestor node's mobility to provide seamless service at any part of the network. In this section, we discuss the user's future location prediction method and based on this information we predict the network quality at that location. We make use of the CSP approach discussed in the previous chapter to find the best fit service composition based

on the predicted information. However, the mobility is a vast area of research and have many attributes such as device handoff, network load balancing, routing protocols, etc. We made the following assumptions based on [5], [19], [62], and [64] that SDN framework used in our approach will adhere such as:

- All the infrastructure devices such as the router and mobile devices are connected.
- The device hand-off between access points are managed by the SDN framework.
- SDN framework manages the network load balancing, scalable control, and fault tolerance.
- SDN control layer manages routing protocols.
- SDN controller keeps track of all the devices connected to it through a periodic status update about the location of the device.
- SDN controller communicates with other controllers through east/west API to share network view and device related information.

4.1.1 User Location Prediction Method

In the mobile scenario, we face the fact that the users are in constant motion thus it is a challenge to provide the service considering the mobility of the user. As both requester and service provider nodes are always on the constant move, the network latency for transmitting input/output data for services varies depending on their location. Thus, our mobility approach consists of two parts: the user's path and the quality of mobile network on the path.

Definition 4 (User's Path) *A user's path is modeled as a triple up = (Time, Position, MF), where:*

- *Time is the time span during which the user is moving. It includes a set of continuous time point.*
- *Position is the set of the user's position corresponding to all the point in the time.*
- *MF is the function that maps time points to the user's location on the path of movement. MF: Time $\longrightarrow$ Position.*

Definition 5 (Network Quality (NQ)) *The quality of the mobile network usually describes the mobile signal strength at a specific location. In this chapter, we mainly consider the data transmission rate as the quality of the mobile network. The function [P] is used to map location to NQ, P: Position $\longrightarrow$ NQ*

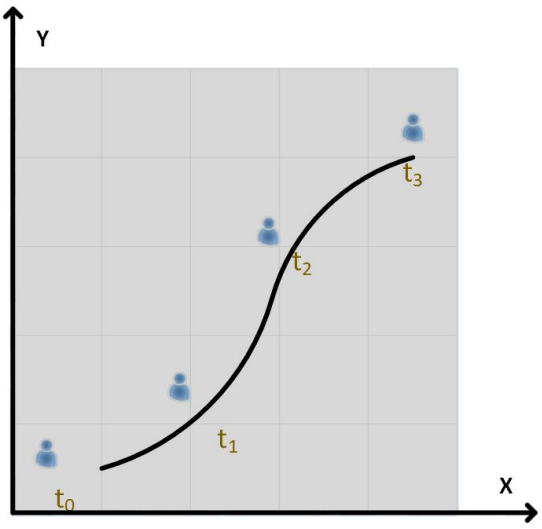

Figure 4.1: Mobility of the user in 2-D

Figure 4.1 shows the pattern in which the requestor node may move. The controller has the local view of its network and the location of the devices in the network. Hence,

we can find the position of requestor node at any specific time point via function MF. The controller can find the time when requestor node starts requesting for service. For example, requestor node requires the computational service at t_0 and receive the composed service at t_1. Meanwhile, the controller knows the location of the requestor node at t_1, and it can obtain the QoS-aware service composition at the corresponding position.

We now discuss how the controller can achieve the user's mobility information. There are three main ways to obtain the moving path of the mobile users. First, the usual route of a mobile user can be recorded by some application/services based on the historical behavior. For example, the user's path from home to work may not change frequently, then such mobility information (geographic path, duration, speed, etc.) can be an estimate based on the recorded historical data. Alternatively, mobility prediction can be one case to obtain the moving path of the mobile user [8]. For example, a Wi-Fi access point service can log the time instances that mobile nodes associate/disassociate with it. Such information can be used to track the mobility of a particular mobile node. Besides, a service requester node may volunteer to report its estimated moving path to our controller. The incentive for a service requester node to disclose such information is the potential to save time and money on mobile connection charges. We note that the similar incentive mechanisms have been studied in mobile ad-hoc networks to achieve cooperative routing and forwarding [36].

However, there is another long-standing challenge in the mobility-aware service composition is that who will be responsible for maintaining all the information about the devices and their mobility while keeping the response time low. Our framework uses SDN as a backbone architecture and the control plane maintains its local view and can further maintain the above information. It can further be enhanced to have more advanced services integrated to it. Firstly, the position information of mobile users can

be monitored by GPS or other location-based service provided by the telecom services providers. Additionally, NQ knowledge can be obtained by monitoring the network as control plane does it frequently. Thus, SDN framework will be able to maintain the proposed architecture.

4.2 Mobility-aware QoS

In this section, we first introduce the mobility-aware QoS function based on the user's path and networks quality mentioned in the previous section. Now we discuss how to compute the mobility-aware QoS of the composed service.

Assumption 1 *The NQ doesn't change during the time each task being transmitted.*

The NQ does vary during the data transmission. However, the variation is not apparent and has limited impact on the time spent on data transmission. Specifically, the execution capability of the mobile device is in GHz, and data volume usually is within several KB, and the transmission data rate is generally in Mbps. So, the complete process takes only few seconds. A mobile user will just move several meters in seconds and the coverage of single base station can reach kilometers even multiple Wi-Fi based infrastructure has a range of several hundred meters. Thus, there is no apparent variation of signal strength across several meters. Therefore, proposed Assumption 1 is to ease the presentation.

However, NQ has the significant role in the selection of a potential service for the service composition. NQ affects the service response time since network quality changes with the mobility of the users. We formulate the Service Response Time (SRT) for service S_i as follows:

$$SRT_{S_i} = \tau_{N_j} \tag{4.1}$$

where τ_{N_j} is the total time required for the execution of the service which includes the time required to send and receive the data, and computation time. τ_{N_j} is calculated similar to the Equation 4.2 in Section 3.5.4, which is provided below for the reference:

$$\tau_{N_j} = \frac{CD^{CC}_{N_j}}{\Gamma_{N_j}} + \frac{CD^{in}_{N_j}}{NQ^{in}_j} + \frac{CD^{out}_{N_j}}{NQ^{out}_j}$$

where $CD^{CC}_{N_j}$ is the CPU cycle required by computational service. Γ_{d_j} is the computation capability (CPU cycles per second) of N_j. NQ^{out}_j is the output data rate and NQ^{in}_j is the input data rate. $CD^{in}_{N_j}$ and $CD^{out}_{N_j}$ are input and output data sizes respectively. Based on Assumption 1, NQ^{in}_j and NQ^{out}_j doesn't change. Once we calculate the service response time, we can further normalize the service response time into service response quality which is defined below.

Definition 6 (Service Response Quality) *SRQ describes the performance of a component service regarding response time in a potential mobility-aware service composition. SRQ is a normalized form of SRT. We calculate the normalized value of SRT as SRQ for each service S_i taking part in composition as follows:*

$$SRQ_{S_i} = 1 - \frac{SRT_{S_i} - min(SRT_S)}{max(SRT_S) - min(SRT_S)} \tag{4.2}$$

The output of the Equation 4.2 is a normalized value of SRT in the range of [0,1] where closer the value to 1 better the response quality.

Definition 7 (Mobility-aware QoS) *Mobility-aware QoS function is the measurement of the overall system performance considering the mobility of the user. For the quantitative measures of the QoS, we consider latency and the degree of match between the requirement of a user's task and composed service, taking into account the user preference and availability of each service taking part in the composition. The mobility-aware quality of service QoS_{MA} is calculated as follows:*

$$QoS_{MA} = max\Big\{W_{SA} \cdot \sum_{j} W_{PS_j} \cdot \Big\{Sim(S_{j,MA}, S_{j,T_{Req}}) + SRQ_{S_j}\Big\}\Big\}, \tag{4.3}$$

$$W_{SA,S_j} = \det(UW_{S_j}, CS_{S_j}, G_{S_j}, ...), \quad where \quad S_j \in S$$

$$W_{SA} = min\Big\{W_{SA,S_j}\Big\}$$

where similarity function Sim is used to measure the similarity between the service S_j in composition and T_{Req}. Similarly, added to similarity output is SRQ_{S_j} which determines the service response quality. However, the output of similarity function and service response quality is between [0,1] which is a normalized output considering the negative and positive both, where the closer the value to the one, the higher the degree of similarity and better the service response quality. The calculation of the similarity measure iterates for all the services and each applied with the user-defined weight W_{PS} given to specific service. We then perform summation and apply the service availability weight W_{SA} calculated by Equation 3.4. Maximum among all the values is our QoS_{MA}.

The QoS_{MA} model is similar to QoS_{SC} model discussed in Section 3.5.1. However, the proposed model is integrated with SRQ. To calculate the mobility-aware QoS of any application which requires a service S from node N, we first need the user time point

(utp) when the user starts to send the data for computation. User's location at the time point utp can be found through the function MF: $Position = MF(utp)$. The quality of the mobile network at that position is derived from the function $P : NQ = P(Position)$. Based on user's path and network quality we define the problem that is to find the best-ranked service for service composition which satisfies both user requirements and network quality equally.

4.3 Mobility-aware CSP Approach

The mobility-aware service composition allows the controller to form the best possible composition of service at the time of need concerning the network quality and user's service preference. Moreover, this approach is the advancement to the CSP approach discussed in Chapter 3 as it follows the same constraints to solve the service composition problem using CSP. We have not made any changes in the CSP approach as well as the procedure. However, mobility-aware approach develops the ranking based on two constraints: the degree of match between the requested service and service composed, and the response quality of the service offered by a node.

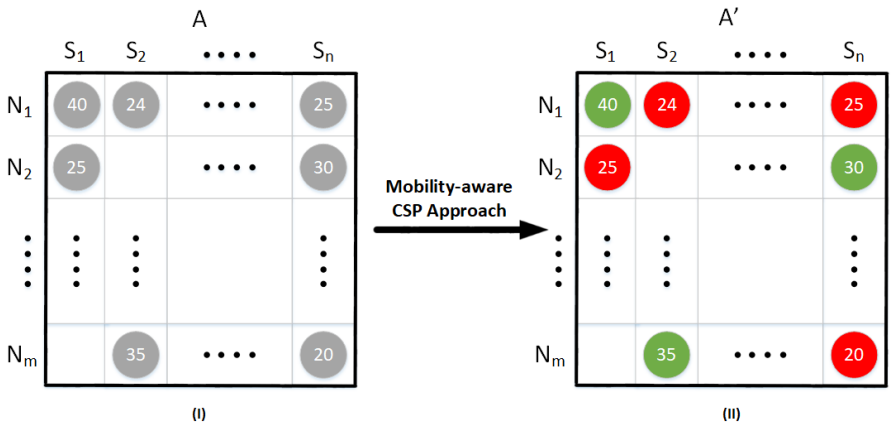

Figure 4.2: Transformation from Matrix A to Matrix A'. Matrix A' shows the Abstraction of Mobility-aware Service Composition.

We describe the mobility-aware service composition problem same as the service composition problem in Section 3.6. Figure 4.2 provides $m \times n$ matrix representing the mobility-aware service composition problem. Matrix A depicts all the possible nodes with ranks that offers a service and Matrix A' depicts the selected node with highest ranking satisfying all the constraints discussed in Section 3.5. The following section describes the various steps of mobility-aware CSP approach.

4.3.1 Location-based Node Selection

The formation of the location-based node table is the first phase of the CSP approach. It is similar to the node selection table mentioned in the previous chapter, as it allows the creation of the Matrix A (Figure 4.2(I)). When there is a requirement of the task execution by a node at t_0, the respective node sends service composition request ($S_{Request}$) to the respective SDN controller for the possible service composition at t_1. Since the requestor node is in constant motion, controller should predict node's future location to provide the service at that next position at t_1, which we represent as *utp*. User's location at the time point *utp* can be found through the function MF: $Position = MF(utp)$. The quality of the mobile network at a position is derived by the controller from the function $P : NQ = P(Position)$. $S_{Request}$ contains T_{Req} which used by the SDN controller as specification list to form the service and decide whether an arrangement is feasible or not. The SDN controller scans its stored device information table and task executed table. Once the SDN controller has investigated all the potential candidates for the service composition, it creates the node table containing device addresses and a unique identifier of all the potential service provider node at the *utp*.

4.3.2 Mobility-aware Service Ranking

The node table generated by SDN controller implies the participation of all the potential devices in the CSP approach. A mobility-aware service selection ranking is developed using the principle Equation 4.3, where a similarity function is computed between service S_j and the equivalent service requirement specified in T_{Req} added with SRQ_{S_j}. Applied to the similarity function are the availability and user-specific/default weights W_{SA,S_j} and W_{PS_j} respectively, of the service S_j. The mobility-aware rank R_{MA} is computed as,

$$R_{MA} = W_{SA,S_j} \cdot W_{PS_j} \cdot \Big\{ Sim(S_j, S_{j,T_{Req}}) + SRQ_{S_j} \Big\}, \tag{4.4}$$

Furthermore, there are no changes made in the service selection algorithm, and service integration and execution phase mentioned in the previous chapter. Upon receiving the node table with rank according to QoS criteria and now mobility integrated to it, we apply the service selection algorithm based on the backtracking forward correction algorithm described in Section 3.6.1.3.

4.4 Evaluation and Analysis

Integration of mobility-aware service composition protocol in MEC environment using SDN has been simulated. We evaluated the performance of proposed adaptive strategy using randomly generated data set according to the Table 4.1. For each participant, we generate random positions regarding x and y coordinate confined in an area of 100*100 m^2. Each of the nodes has the maximum upload transmission rate of 20 Mbps between requestor node and service provider node. However, download transmission rate is not considered for simulation as we assume that the output data is decidedly less in size to get impacted by download transmission rate. We have used MATLAB and JAVA SDK

Table 4.1: Simulation Parameter Summary

Duration	30 service requests
Number of controllers	1
Number of nodes	50
Number of hop	1
Service composition length	5
Computation power of nodes	0.5 GHz, 0.8 GHz and 1.0 GHz
Computation Data	5000 KB
CPU cycle required for Computation	1000 Megacycles
2-D grid size	100 * 100 m^2
Position of nodes	Randomly distributed
Upload transmission rate	20 Mbps

for all simulations and experiments. We have generated random future locations of the users for different user time points. The scheme and methodology of the simulation are same as mentioned in the previous section and the only process changed is the selection of the node in the method.

4.4.1 Result and Analysis

Results shown in this section represent the task execution time in the simulation environment. We compare our result with the state of the art result of Energy-Saving Greedy Algorithm (EGA) and Conditional Optimal Algorithm (COA) described by Yong and Song et al. [13]. We used EGA and COA to compose service for our model and use as a benchmark solution. We altered both COA and EGA according to the mobile environment. We have further used a random scheduling algorithm also as a performance benchmark, where the task offloading decision is a random permutation of the candidate selection with mobility induced to nodes.

4.4.2 Effect of Data Size on Response time

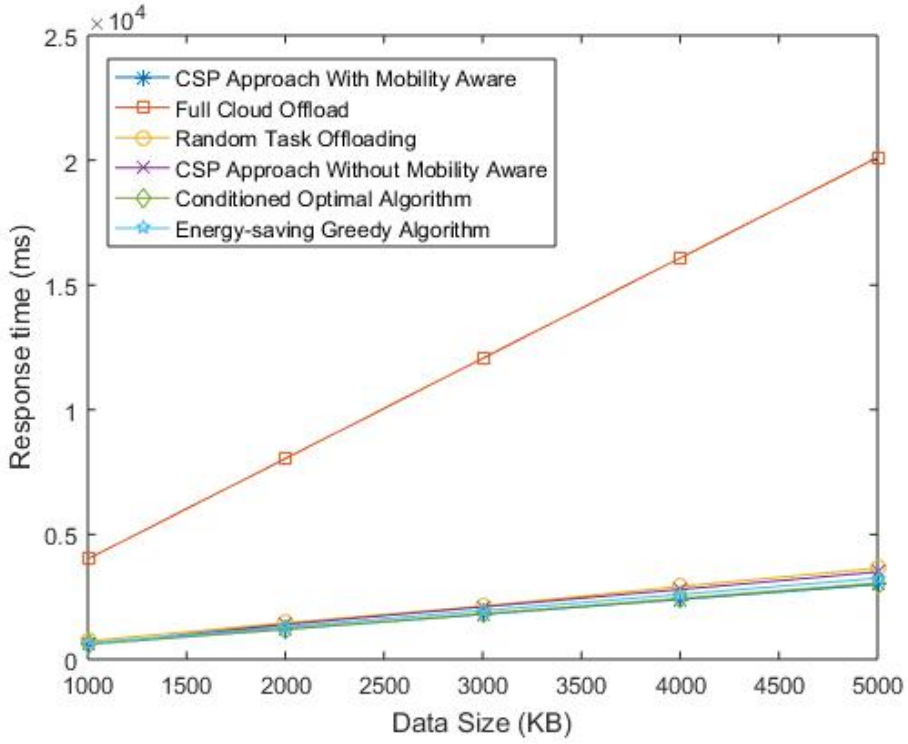

Figure 4.3: Response Time vs. Data Size

Figure 4.3 depicts the relation between the various Data size and Response time. It shows that the response time increases with increase in offloading Data size. COA and EGA both are efficient energy-based approach thus our approach provides lower response time. The results show that the there is a significant decrease in response time when compared to cloud execution, random task offloading algorithm, COA and EGA. This is because the CSP approach selects the nodes based on the requirement and the selected nodes are the best nodes available in the network with optimal response time. However, in the random selection process, the arbitrary nodes are selected. Cloud execution always has the higher response time since there is an extra time involved in offloading and downloading the data as compared to the peer node. The CSP approach

with the mobility-aware mechanism selects the node based on user's future location and signal strength. Thus making it more efficient regarding the response time.

4.4.3 Impact of Variation of Signal Strength

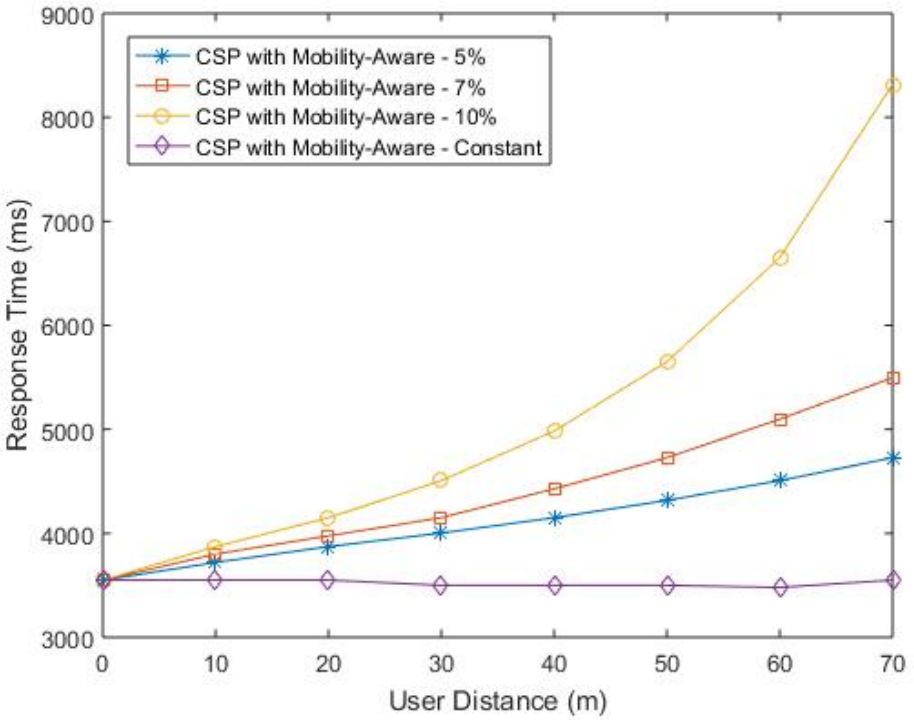

Figure 4.4: Response Time vs. User Distance with 5%, 10% and, 15% degrade in signal strengths and constant signal strength.

In the proposed mobility model, the two functions MF and P determine the relation between user's movement and variation of signal strength. For our experiments, we have combined the user movement function $Position = MF(time)$ and mobility strength function $NQ = P(Position)$ in one function of signal strength: $NQ = G(time) = P(MF(time)$, which gives the signal strength with the increase in time. Since many factors are affecting the signal strength (such as a user's speed, distance from signal stations, properties of each signal station, etc.), the variation of signal strength in practice

cannot be defined by any function, and it is difficult to acquire the real data for signal strength. Therefore, we assume four different cases of signal strength to simulate the variation of signal strength during user's movement. First, we simulated with the constant signal strength which provides an ideal case where there is no net loss and the net gain in signal strength during the movement. Secondly, with 5%, 7%, and 10% signal strengths which shows the degrade in signal strength during the movement. Therefore, this helps us to simulate the scenario where an increase in speed degrades the signal strength. User Distance depicts the different point in future with relatively far from each other at each point. Figure 4.4 shows the behavior of our proposed method with different signal strengths.

4.4.4 Effect of Increasing User Distance on Response Time

Figure 4.5 shows the effectiveness of our approach concerning the response time with the increase in distance from the initial position. During the simulation at each time point in future, the user has made the requests for service execution. The graph evidently shows that our approach is readily making the service available at any point in time with minimal difference in response time. Response time in mobility-aware method is stable, and it is because our approach makes the future prediction of user's environment and obtain the best possible nodes keeping the signal strength in the notice. CSP approach without mobility-aware mechanism evidently is not able to predict the future position thus with the increase in distance decrease the signal strength of the composed node which in turn increase the response time. The random approach is also not able to select appropriate node concerning user's future position.

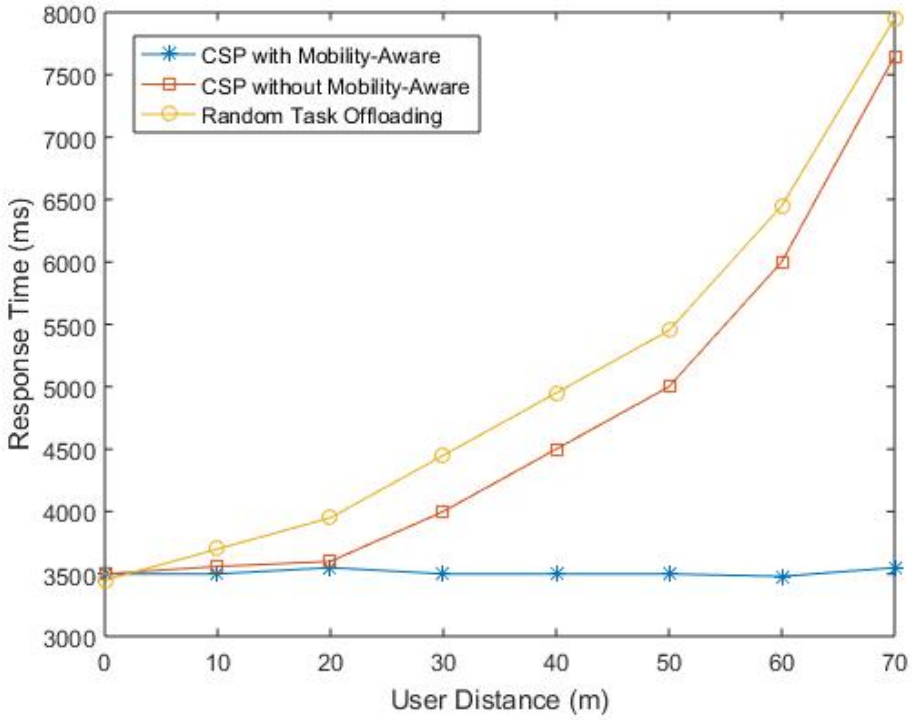

Figure 4.5: Response Time vs. User Distance

4.5 Conclusion

We provided a mobility-aware approach for software-defined service composition where the selection of each possible device for service composition is based on the future location of devices. Our work is the first work to the best of our knowledge for composing service using SDN framework with a mobility-aware mechanism. We also considered the signal strength of each node concerning their future location to get the desired devices for composition. The experimental simulation result shows that our approach can obtain better response time in the mobile environment compared to the random offloading, conditioned and greedy algorithms. One of the main achievements is that our method stabilizes the response time by selecting the best nodes at the future locations when user's mobility causes variation in signal strength.

Chapter 5

Contribution and Future Work

5.1 Contribution

The focus of the dissertation research has been the development of an SDN-based service composition model. We have suggested a distinct approach for solving the computation offloading problem using service composition means. We have provided a better strategy for service and device discovery which can reduce the execution time by finding the similarities between requested service and potential service in composition. Our method proves to be a user-centric approach and better for moderate and large data size of the computational task.

The approach to select the optimal devices based on energy consumption in service compositions are not always suitable especially when the users have a specific service preference. It is not adequate to use the greedy or random approach to satisfy the user constraints. In lieu of that concern, we proposed CSP approach that helps in saving a lot of time and experimental results validate the efficiency of the approach.

Apart from this, we have provided a mobility-aware service composition approach where user's mobility is considered as the necessary constraint in selecting the device for

service composition. Moreover, our work is first in our knowledge which devises an SDN-based mobility-aware service composition for computational offloading. The mobility-aware service composition mechanism considers the future location of the mobile devices while composing the service. In addition to this, all the proposed approaches consider QoS while performing the composition. Results show that our method can match user's requirement by up to 98%.

5.2 Future Work

We have discussed on predicting the future locations of the devices for service composition and assumed that the signal strength and speed of the devices does not change during the execution of the tasks. In reality, the devices will move even during the execution of the tasks. Therefore, we plan to extend our work to develop the mobility management which will maintain the low latency during the execution of the task and makes sure that the sudden disconnection of a device does not affect the overall data execution in the network. We have also considered the independent services for experiments. However, in future, we tend to focus on finding an optimal procedure for the dependent tasks and maintain the low response time of the composed service and satisfy user's requirement.

Computational offloading decisions and service composition may degrade the overall performance of the SDN controller. We plan to extend our work to minimize the overhead time on SDN controller by using distributed approach. Since we have worked on mobility-aware approach to find the optimal nodes, we further plan to extend our work to include bandwidth-aware mechanism which will make the SDN-based service composition, a bandwidth conscious mechanism as well.

Privacy and security are one of the primary concerns in MEC. We plan to induce the efficient mechanism to make the system and participating devices more secure, explicitly

in terms of user data. In addition to this, there is also a possibility that a device or a group of devices get compromised during the task execution. This will lead to the inconsistency in computed data. For this, it is essential to detect attackers, and this can easily be extended to the existing framework based on the information uploaded by all the node in the network.

References

[1] Sugam Agarwal, Murali Kodialam, and TV Lakshman. Traffic Engineering in Software-Defined Networks. In *Proceedings of INFOCOM*, pages 2211–2219. IEEE, 2013.

[2] Arif Ahmed and Ejaz Ahmed. A Survey on Mobile Edge Computing. In *10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–8. IEEE, 2016.

[3] Mohamed Ahmed, Felipe Huici, and Armin Jahanpanah. Enabling Dynamic Network Processing with ClickOS. In *Proceedings of the SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 293–294. ACM, 2012.

[4] Chang-Woo Ahn and Sang-Hwa Chung. SDN-Based Mobile Data Offloading Scheme Using a Femtocell and WiFi Networks. *Mobile Information Systems*, pages 1–15, 2017.

[5] Hassan Ali-Ahmad, Claudio Cicconetti, Antonio De la Oliva, Vincenzo Mancuso, Malla Reddy Sama, Pierrick Seite, and Sivasothy Shanmugalingam. An SDN-Based Network Architecture for Extremely Dense Wireless Networks. In *SDN for Future Networks and Services (SDN4FNS)*, pages 1–7. IEEE, 2013.

[6] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. ONOS: Towards an Open, Distributed SDN OS. In *Proceedings of the 3rd Workshop on Hot Topics in Software-Defined Networking*, pages 1–6. ACM, 2014.

[7] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog Computing: A Platform for Internet of Things and Analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, pages 169–186. Springer, 2014.

[8] Francesco Calabrese, Giusy Di Lorenzo, and Carlo Ratti. Human Mobility Prediction Based on Individual and Collective Geographical Preferences. In *13th International Conference on Intelligent Transportation Systems (ITSC)*, pages 312–317. IEEE, 2010.

[9] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking Control of the Enterprise. In *SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.

[10] Fabio Casati, Ski Ilnicki, LiJie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. Adaptive and Dynamic Service Composition in eFlow. In *International Conference on Advanced Information Systems Engineering*, pages 13–31. Springer, 2000.

[11] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient Multi-user Computation Offloading for Mobile-Edge Cloud Computing. *ACM Transactions on Networking*, 24(5):2795–2808, 2016.

[12] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: Making Smartphones Last

Longer with Code Offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 49–62. ACM, 2010.

[13] Yong Cui, Jian Song, Kui Ren, Minming Li, Zongpeng Li, Qingmei Ren, and Yangjun Zhang. Software Defined Cooperative Offloading for Mobile Cloudlets. *ACM Transactions on Networking*, 2017.

[14] Sreekanth Dama, Valin Sathya, Kiran Kuchi, and Thomas Valerrian Pasca. A Feasible Cellular Internet of Things: Enabling Edge Computing and the IoT in Dense Futuristic Cellular Networks. *Consumer Electronics Magazine*, 6(1):66–72, 2017.

[15] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[16] Thinh Quang Dinh, Jianhua Tang, Quang Duy La, and Tony QS Quek. Adaptive Computation Scaling and Task Offloading in Mobile Edge Computing. In *Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2017.

[17] Avri Doria, J Hadi Salim, Robert Haas, Horzmud Khosravi, Weiming Wang, Ligang Dong, Ram Gopal, and Joel Halpern. Forwarding and Control Element Separation (ForCES) Protocol Specification. *Internet Engineering Task Force (IETF)*, 2010.

[18] David Erickson. The Beacon Openflow Controller. In *Proceedings of the Second SIGCOMM Workshop on Hot Topics in Software-Defined Networking*, pages 13–18. ACM, 2013.

[19] Fabio Giust, Luca Cominardi, and Carlos J Bernardos. Distributed Mobility Management for Future 5G Networks: Overview and Analysis of Existing Approaches. *Communications Magazine*, 53(1):142–149, 2015.

[20] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: Towards An Operating System for Networks. *SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[21] Harshit Gupta, Shubha Brata Nath, Sandip Chakraborty, and Soumya K Ghosh. SDFog: A Software Defined Computing Architecture for QoS Aware Service Orchestration Over Edge Devices. *arXiv Preprint arXiv:1609.01190*, 2016.

[22] J Octavio Gutierrez-Garcia and Kwang-Mong Sim. Self-Organizing Agents for Service Composition in Cloud Computing. In *Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 59–66. IEEE, 2010.

[23] Karim Habak, Mostafa Ammar, Khaled A Harras, and Ellen Zegura. Femto Clouds: Leveraging Mobile Devices to provide Cloud Service at the Edge. In *8th International Conference on Cloud Computing (CLOUD)*, pages 9–16. IEEE, 2015.

[24] Nikhil Handigol, Mario Flajslik, Srini Seetharaman, Nick McKeown, and Ramesh Johari. Aster* x: Load-Balancing as a Network Primitive. In *9th GENI Engineering Conference (Plenary)*, pages 1–2, 2010.

[25] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In *Proceedings of the First Workshop on Hot Topics in Software-Defined Networks*, pages 19–24. ACM, 2012.

[26] Kalapriya Kannan and Subhasis Banerjee. Scissors: Dealing with Header Redundancies in Data Centers through SDN. In *8th International Conference and Workshop on Systems Virtualiztion Management (SVM) Network and Service Management (CNSM)*, pages 295–301. IEEE, 2012.

[27] Pradeeban Kathiravelu, Tihana Galinac Grbac, and Luís Veiga. A FIRM Approach for Software-Defined Service Composition. In *39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 565–570. IEEE, 2016.

[28] Hyojoon Kim and Nick Feamster. Improving Network Management with Software-Defined Networking. *Communications Magazine*, 51(2):114–119, 2013.

[29] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A Distributed Control Platform for Large-Scale Production Networks. In *OSDI*, volume 10, pages 1–6, 2010.

[30] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading. In *Proceedings of INFOCOM*, pages 945–953. IEEE, 2012.

[31] Karthik Kumar and Yung-Hsiang Lu. Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? *Computer*, 43(4):51–56, 2010.

[32] Vipin Kumar. Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, 13(1):32, 1992.

[33] JL Le Roux. Path Computation Element (PCE) Communication Protocol (PCEP). *Internet Engineering Task Force (IETF)*, 2009.

[34] Kyunghan Lee, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. Mobile Data Offloading: How Much can WiFi Deliver? *ACM Transactions on Networking (TON)*, 21(2):536–550, 2013.

[35] Dan Levin, Andreas Wundsam, Brandon Heller, Nikhil Handigol, and Anja Feldmann. Logically Centralized?: State Distribution Trade-offs in Software Defined Networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, pages 1–6. ACM, 2012.

[36] Ze Li and Haiying Shen. Game-Theoretic Analysis of Cooperation Incentive Strategies in Mobile Ad-hoc Networks. *Transactions on Mobile Computing*, 11(8):1287–1303, 2012.

[37] Yang Liu, Changqiao Xu, Yufeng Zhan, Zhixin Liu, Jianfeng Guan, and Hongke Zhang. Incentive Mechanism For Computation Offloading Using Edge Computing: A Stackelberg Game Approach. *Computer Networks*, 129:399–409, 2017.

[38] Tom H Luan, Longxiang Gao, Zhi Li, Yang Xiang, Guiyi Wei, and Limin Sun. Fog Computing: Focusing on Mobile Users at the Edge. *arXiv Preprint arXiv:1502.01815*, 2015.

[39] Yan Luo, Pablo Cascon, Eric Murray, and Julio Ortega. Accelerating OpenFlow Switching with Network Processors. In *Proceedings of the 5th Symposium on Architectures for Networking and Communications Systems*, pages 70–71. ACM/IEEE, 2009.

[40] Pavel Mach and Zdenek Becvar. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[41] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. Fog Computing: A Taxonomy, Survey and Future Directions. *Internet of Everything*, pages 103–130, 2018.

[42] Nick McKeown. How SDN will Shape Networking. *Open Networking Summit*, 2011.

[43] Madjid Merabti, Paul Fergus, Omar Abuelma'atti, Heather Yu, and Charlie Judice. Managing Distributed Networked Appliances in Home Networks. *In Proceedings of the IEEE*, 96(1):166–185, 2008.

[44] Antti P Miettinen and Jukka K Nurminen. Energy Efficiency of Mobile Clients in Cloud Computing. *HotCloud*, 10:4–4, 2010.

[45] Eugene Ng, Z Cai, and AL Cox. Maestro: A System for Scalable Openflow Control. *Rice University, Houston, TX, USA, TSEN Maestro-Techn. Rep, TR10-08*, 2010.

[46] Van-Giang Nguyen, Truong-Xuan Do, and YoungHan Kim. SDN and Virtualization-Based LTE Mobile Network Architectures: A Comprehensive Survey. *Wireless Personal Communications*, 86(3):1401–1438, 2016.

[47] Kevin Phemius, Mathieu Bouet, and Jérémie Leguay. Disco: Distributed Multi-Domain SDN Controllers. In *Network Operations and Management Symposium (NOMS)*, pages 1–4. IEEE, 2014.

[48] Kiran K Rachuri, Cecilia Mascolo, Mirco Musolesi, and Peter J Rentfrow. Sociablesense: Exploring the Trade-offs of Adaptive Sampling and Computation Offloading for Social Sensing. In *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, pages 73–84. ACM, 2011.

[49] Lakshmana Kumar Ramasamy and MS Irfan Ahmed. SWEFT: Semantic Web Service Engine for Telecommunication-An Automatic Discovery and Composition Through Genetic Approach. *Science and Technology International Journal of u-and e-Service*, 9(9):139–154, 2016.

[50] Pablo Rodriguez-Mier, Carlos Pedrinaci, Manuel Lama, and Manuel Mucientes. An Integrated Semantic Web Service Discovery and Composition Framework. *Transactions on Services Computing*, 9(4):537–550, 2016.

[51] Koya Sato and Takeo Fujii. Radio Environment Aware Computation Offloading with Multiple Mobile Edge Computing Servers. In *Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 1–5. IEEE, 2017.

[52] Usman Shaukat, Ejaz Ahmed, Zahid Anwar, and Feng Xia. Cloudlet Deployment in Local Wireless Networks: Motivation, Architectures, Applications, and Open Challenges. *Journal of Network and Computer Applications*, 62:18–40, 2016.

[53] Scott Shenker, Martin Casado, Teemu Koponen, Nick McKeown, et al. The Future of Networking, and The Past of Protocols. *Open Networking Summit*, 20:1–30, 2011.

[54] Helmut Simonis. Sudoku as A Constraint Problem. *Proc. 4th Int. Works. Modelling and Reformulating Constraint Satisfaction Problems*, pages 13–27, 2005.

[55] Andrey Somov and Raffaele Giaffreda. Powering IoT Devices: Technologies and Opportunities. *Newsletter*, 2014.

[56] Tolga Soyata, Rajani Muraleedharan, Colin Funai, Minseok Kwon, and Wendi Heinzelman. Cloud-Vision: Real-time Face Recognition using a Mobile-Cloudlet-Cloud Acceleration architecture. In *Symposium on Computers and Communications (ISCC)*, pages 59–66. IEEE, 2012.

[57] OpenFlow Switch Specification. Open Networking Foundation. *Version ONF TS-015*, 1(3):1–164, 2013.

[58] Neil T Spring and David Wetherall. A Protocol-Independent Technique For Eliminating Redundant Network Traffic. *SIGCOMM Computer Communication Review*, 30(4):87–95, 2000.

[59] Tarik Taleb, Sunny Dutta, Adlen Ksentini, Muddesar Iqbal, and Hannu Flinck. Mobile Edge Computing Potential in Making Cities Smarter. *Communications Magazine*, 55(3):38–43, 2017.

[60] Tarik Taleb and Adlen Ksentini. An Analytical Model for Follow Me Cloud. In *Global Communications Conference (GLOBECOM)*, pages 1291–1296. IEEE, 2013.

[61] Amin Tootoonchian and Yashar Ganjali. HyperFlow: A Distributed Control Plane for OpenFlow. In *Proceedings of the Internet Network Management Conference on Research on Enterprise Networking*, pages 1–6, 2010.

[62] Luca Valtulina, Morteza Karimzadeh, Georgios Karagiannis, Geert Heijenk, and Aiko Pras. Performance Evaluation of a SDN/OpenFlow-based Distributed Mobility Management (DMM) approach in virtualized LTE systems. In *Globecom Workshops (GC Wkshps)*, pages 18–23. IEEE, 2014.

[63] Guohui Wang, TS Ng, and Anees Shaikh. Programming Your Network at Run-Time for Big Data Applications. In *Proceedings of the First Workshop on Hot Topics in Software-Defined Networks*, pages 103–108. ACM, 2012.

[64] Di Wu, Dmitri I Arkhipov, Eskindir Asmare, Zhijing Qin, and Julie A McCann. UbiFlow: Mobility Management in Urban-Scale Software-Defined IoT. In *Conference on Computer Communications (INFOCOM)*, pages 208–216. IEEE, 2015.

[65] Junjie Xie, Deke Guo, Zhiyao Hu, Ting Qu, and Pin Lv. Control Plane of Software Defined Networks: A Survey. *Computer Communications*, 67:1–10, 2015.

[66] Minlan Yu, Lavanya Jose, and Rui Miao. Software-Defined Traffic Measurement with OpenSketch. In *NSDI*, volume 13, pages 29–42, 2013.

[67] Yuan Zhang, Hao Liu, Lei Jiao, and Xiaoming Fu. To Offload or Not to Offload: An Efficient Code Partition Algorithm for Mobile Cloud Computing. In *1st International Conference on Cloud Networking (CLOUDNET)*, pages 80–86. IEEE, 2012.