MOBILE EDGE CLOUD ARCHITECTURE FOR FUTURE LOW-LATENCY APPLICATIONS

BY SUMIT MAHESHWARI

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Dipankar Raychaudhuri

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

May, 2020

ABSTRACT OF THE DISSERTATION

Mobile Edge Cloud Architecture for Future Low-latency Applications

by Sumit Maheshwari

Dissertation Director: Dipankar Raychaudhuri

This thesis presents the architecture, design, and evaluation of the mobile edge cloud (MEC) system aimed at supporting future low-latency applications. Mobile edge clouds have emerged as a solution for providing low latency services in future generations (5G and beyond) of mobile networks, which are expected to support a variety of real-time applications such as AR/VR (Augmented/Virtual Reality), autonomous vehicles and robotics. Conventional cloud computing implemented at distant large-scale data centers incurs irreducible propagation delays of the order of 50-100ms or more that may be acceptable for current applications but may not be able to support emerging real-time needs. Edge clouds considered here promise to meet the stringent latency requirements of emerging classes of real-time applications by bringing compute, storage, and networking resources closer to user devices. However, edge clouds are intrinsically local and have a smaller scale and are thus subject to significantly larger fluctuations in offered traffic due to factors such as correlated events and user mobility. In addition, edge computing systems by definition are distributed across multiple edge networks and hence are associated with considerable heterogeneity in bandwidth and compute resources. Considering these challenges, this thesis analyzes the requirements posed by the edge clouds and proposes specific techniques for control, network routing, data

migration, and dynamic resource assignment which can be employed to support low-latency applications.

The thesis starts by analyzing system-level edge cloud requirements for low-latency by deploying a set of sample AR applications, namely, annotation-based assistance, and smart navigation. A city-scale MEC system is analyzed for achievable latency when running AR applications using existing core clouds as well as the proposed distributed edge cloud infrastructure. Performance evaluation results are presented to understand the trade-offs in key system parameters such as core cloud latency and inter-edge or core-to-edge network bandwidth. The results show that while the core cloud-only system outperforms the edge-only system having low inter-edge bandwidth, a distributed edge cloud selection scheme can approach global optimal assignment when the edge has sufficient compute resources and high inter-edge bandwidth. Adding capacity to an existing edge cloud system without increasing the inter-edge bandwidth contributes to network-wide congestion and can reduce system capacity.

Next, a specific network-assisted cloud resource management technique is described that uses the concept of named-object architecture to create a named-object based virtual network (NOVN) inherently supporting application specific routing specifically designed to enable Quality of Service (QoS) in MEC. The results validate the feasibility of the named-object approach, showing minimal VN processing, control overhead, and latency. The results also validate application specific routing (ASR) functionality for an example latency constrained edge cloud service scenario.

Further, user mobility and edge cloud system load balancing are handled by enabling dynamic service migration. Container migration is emerging as a potential solution that enables dynamic resource migration in virtualized networks and mobile edge cloud (MEC) systems. The orchestrated, lightweight container migration model is designed and evaluated for a real-time application (license plate recognition) using performance metrics such as the average system response time and the migration cost for different combinations of load, compute resources, inter-edge cloud bandwidth, network, and application latency. The concept of NOVN and service migration are then applied to the advanced driver assistance systems (ADAS) geared towards autonomous driving using

Augmented Reality (AR). The experiments show that the low-latency ADAS applications with an average system latency of less than 100 ms for the applications can be supported. The key observations from this study are: (1) machine type plays a crucial role in deciding migration, (2) applications requiring higher computation capabilities, for instance, annotation-based assistance should be offloaded to the closest available lightly-loaded edge cloud, (3) the latency of applications requiring pre-fetched data has fewer avenues for optimization, and (4) service migration should consider network bandwidth, system load, and compute capability of the source and the destination.

The work on edge cloud resource assignment and networking motivated the design of a general-purpose control plane that supports the exchange of essential control information (such as compute and network capabilities, current workloads, bandwidth/latency, etc.) between edge cloud domains in a region. The existence of such a control plane enables distributed resource management, application-aware routing, and task assignment algorithms without the requirement of a single point of control. Therefore, the final part of this thesis focuses on creating a lightweight control protocol that can provide neighboring edge clouds with visibility of their computing and network resources along with current load metrics. The proposed design promotes regional awareness of available resources in a heterogeneous multi-tenant environment to enable cooperative techniques such as cluster computing, compute offloading, or service chaining. The design of a specific control plane protocol followed by a system-level evaluation of the performance associated with task assignment and routing algorithms enabled by the framework is presented. The evaluation is based on a prototype system with a heterogeneous network of compute clusters participating in the control plane protocol and executing specified resource sharing algorithms. An application-level evaluation of latency vs. offered load is also carried out for an example time-critical application (image analysis for traffic lane detection) running on the ORBIT testbed confirming that significant performance gains can be achieved through cooperation at the cost of modest complexity and overhead.

Acknowledgements

People who are willing to go above and beyond to support you are difficult to find; I am lucky to have them in abundance in my life. A rather difficult job is to thank them enough for their contributions. I hope these words do justice expressing how indebted I feel.

First and foremost, I wish to extend my deepest gratitude to my advisor Prof. Dipankar Raychaudhuri for believing in me and being my polestar for guiding me through every step on the way. Of all the hard jobs around, one of the hardest is being a good teacher. I could never thank him enough for imparting his technical expertise and life-lessons, and clarifying what I was thinking when it was not apparent to me. He has always given me the best advice I could ever seek, both personally and professionally, and never wavered in his generosity and sharp engagement with my work by providing unique perspectives. His kindness and compassion are synergic with that of his better half, Arundhati aunty. Thank you, aunty, for your love and care.

I would like to thank my proposal and thesis committee members, Dr. N. K. Shankaranarayanan, Prof. Emina Soljanin, Prof. Roy Yates, and Prof. Richard Martin. My external examiner, Shankar gave me many opportunities to observe and learn from his work. I look forward to continuing our collaboration. Emina's teaching and research always inspired me. Thoughtful and generous are the two words I think when I think of her. I am grateful to Roy for bringing the best out of me. His invaluable insights and feedback during the weekly team meetings greatly improved the work. My infinite gratitude to all the committee members.

WINLAB has become an unforgettable place for me; it's hard to imagine WINLAB without Ivan Seskar. From him, I learned how to solve problems from the systems view that gave practical wings to my theories. He imparted deep technical suggestions

during our friendly conversations and kept a tab open for me even in the busiest of his times. I sincerely thank Prof. Zoran Gajic for navigating me through this journey. Thanks are also due to the professors who have shaped me as a student including Prof. Narayan Mandayam, Prof. Predrag Spasojavic, Prof. Gil Zussman, Prof. Gala Yadgar, and Prof. Yanyong Zhang.

This thesis is incomplete without mentioning the support of my mentors, collaborators, and friends – Dr. Francesco, Dr. Jiachen, Shalini, Wuyang, Prasad, Filippo, Anirudh, Kishore, Krishnamohan, Mohit, Aayush, Tingjun, Hanif, Shreyasee, Siddharth, Parishad, Murtadha, Vishakha, Mareesh, Bhargav, Ayman, Prof. Mahapatra, Prof. Chakrabarti, Prof. Ganguly, Dr. Vasu, Suganya, Naveen, Dr. Walid, Sameer, Andrea, Ray Miller, Dr. Abhigyan, and Matthew for all the big and small ways you all have pitched in.

I would like to acknowledge the competent staff members of WINLAB, ECE, and Rutgers including Noreen, Elaine, Christy, Mike, Jake, Lisa, Janice, Elisa, Jenny, and Arletta, for their prompt help.

Deepest heartfelt thanks to my family whose unconditional love and selfless support are with me in whatever I pursue. I could not have imagined doing a Ph.D. without the much-needed love and affection of my wife, Kritika, for being there, always, whatsoever. Thank you for sailing me through the tough times, patiently listening to all the technical jibber-jabber, taking care of our kids when I almost every day said that I am busy today, and boosting my morale during the darkest of the moments. I don't think I can ever thank you enough, and therefore I am dedicating this thesis to you.

Dedication

To my lovely wife Kritika, and wonderful kids Chinmay & Aavya

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The computing requirements of mobile devices are growing due to a rise in newer applications such as Augmented Reality (AR), Virtual Reality (VR), autonomous vehicles, and robotics. The surge in usage of mobile devices (mobile data traffic will increase seven-fold between 2017–2022[2]) has also resulted in tremendous increase of user expectations both for the data rate and the quality of service (QoS). The mobile applications' focus is therefore shifting from *local execution* towards the *always connected* paradigm requiring seamless access to cloud computing, storage, and networking intended to offset limited computation capability and battery life at mobile devices. The concept of Mobile Edge Cloud (MEC; also known as Multi-access Edge Computing) is introduced by placing the compute, storage, and network components close to the users [3, 4] to potentially fulfill extensive computing requirements of emerging low-latency class of applications. Complex application software such as augmented and virtual reality etc., running on these mobile devices require data to be processed in ∼ms, something that mobile devices (even those equipped with accelerator GPUs) cannot keep up with. This motivates the edge cloud architecture in which the network infrastructure helps improve the user's QoE. The number of devices (scalability) and their performance requirements (QoS) to be supported by the MEC architecture bring in additional research challenges yet to be explored. This thesis[†] proposes architecture, design, and evaluation of the MEC system with a specific focus on supporting future low-latency applications. The specific results presented in this thesis include a study of the scalability and performance

[†]The work presented in this thesis is partially funded by NSF Future Internet Architecture - Next Phase (FIA-NP) CNS#1345295, and COSMOS project CNS#1827923 funded by NSF and PAWR Project Office.

of AR applications supported by city-scale MEC system, virtual network techniques to enable QoS support in MEC, handling load balancing and user mobility, and providing decentralized control plane protocol to enable distributed orchestration of edge cloud resources.

Current cloud computing systems are designed to support low-cost and scalable compute offloading which cannot provide a stringent bound on the round trip delays of future low-latency applications[5, 6, 7] mainly due to their distant placement inducing higher network delays, and uncertain bandwidth connectivity between the request endpoint and the cloud [8]. Note that while the shift from infrastructure based computing to cloud computing was based on the economic factors, the shift towards MEC is motivated by low-latency and high bandwidth scenarios that require tighter placement and control of computing and network resources. Although grid computing also shares several common aspects with the MEC including peer to peer computing[9], it cannot fulfill low-latency demands due to its loosely connected compute and network protocols thus creating a lack of availability and trust[10, 11]. While MEC promises to fulfill the application QoS, its geographically local, resource constrained design will not be sufficient to handle load surges. Therefore the inter-MEC, as well as the MEC to cloud offloading schemes, must be used to support AR applications. A detailed study of scalability and performance of such a hybrid MEC system using a city-scale simulation model is thus undertaken in this thesis wherein MECs are complemented with the central cloud. The work presented in this thesis is an early direction to provide system capacity details of hybrid MEC with different resource distributions, load and inter-edge bandwidth which has the potential to guide future MEC deployments.

The geographical distribution of MEC resources leads to challenges of inter-domain connectivity and dynamic re-routing while providing seamless service to a user. Existing L2 solutions such as VLAN [12] and MPLS [13] cannot provide cross-domain connectivity while overlay based L3 solutions such as VINI [14] bring in the additional set-up as well as run-time overhead. MEC being an integral part of 5G networks under consideration [15], should support seamless service integration by connecting users to their respective edge clouds. To achieve this, we propose an in-network, name-object

based [16] dynamic virtual network design (NOVN), which can provide application specific routing of a request to a *best* edge cloud. NOVN enhances the MEC architecture by inherently supporting resource management, network slicing, and compute and network orchestration.

The localized nature of MECs causes high load variability due to user mobility. The prevalent technique to deal with such a scenario is task offloading [17, 18] which introduces additional network path extension delay when using a uniform MEC entry/exit point. The virtualization of computing functions in the form of the virtual machines and more recently the lightweight containers have opened newer avenues of handling user mobility and MEC load by using service migrations which are explored in this thesis. An orchestrated framework to minimize service downtime and improve QoS is designed and evaluated for taxicabs in San Francisco city.

The typical system model for edge clouds is based on the use of a central controller for coordination of compute and network resources. The centralized control model does not apply to a distributed system implemented across multiple edge network domains, possibly involving a multiplicity of service providers. For a distributed edge cloud model to work, a control plane is needed to exchange both network and compute resource information between edge clouds located in a region. Existing MEC architectures such as hierarchical [19, 20] as well as centralized [21] implicitly assume the availability of control plane information. This thesis proposes to design, implement and evaluate the performance of a fully decentralized control plane protocol for MEC using an overlay protocol design, allowing the use of the existing network fabric while ensuring flexibility, simplicity, and elasticity [22]. Overlay solutions are a popular choice in the literature due to their implementation simplicity and compatibility advantages [23, 24]. A common solution for distributing inter-network information is the Border Gateway Protocol (BGP) which disseminates the IP routes between the Autonomous Systems (ASes) by forming peering relationships [25]. However BGP by itself is designed only for routing and cannot provide updates with the short time constants associated with edge cloud resources. Overlay based large-scale testbeds such as PlanetLab and GENI [26, 27] are configured for compute and network resources using resource specifications (RSpec) [28]

which enables a full description of the network, control of the network topologies, and network-aware resource placement. Our protocol design complements these testbeds for run-time resource discovery extending the current static configuration capabilities with Rspec. The edge cloud network is a shared infrastructure system in which networking entities such as routers play a significant role in MEC's performance. Therefore, supporting the views of [29, 30] we believe that a tightly coupled network participation, for instance, routers disseminating network state information to the MEC nodes helps to improve the performance and scalability.

A general technology solution for edge clouds will thus require distributed control algorithms and associated control plane protocols to realize a closed loop, low-latency control as well as the data plane for compute, storage and networking. The control plane information thus obtained can be used in a multitude of applications such as cluster computing, service migration, and service chaining, thereby motivates us to design and develop an end-to-end framework to support future applications.

1.1 Organization of the thesis

Chapter 2 presents an analysis of the scalability and performance of an edge cloud system designed to support latency-sensitive applications. A system model for geographically dispersed edge clouds is developed by considering an urban area such as Chicago and co-locating edge computing clusters with known Wi-Fi access point locations. The model also allows for provisioning of network bandwidth and processing resources with specified parameters in both edge and the cloud. The model can then be used to determine application response time (sum of network delay, compute queuing and compute processing time), as a function of offered load for different values of edge and core compute resources, and network bandwidth parameters. Numerical results are given for the city-scale scenario under consideration to show key system-level trade-offs between edge cloud and conventional cloud computing. Alternative strategies for routing service requests to edge vs. core cloud clusters are discussed and evaluated. Key conclusions from the study are: (a) the core cloud-only system outperforms the edge-only system having low inter-edge bandwidth, (b) a distributed edge cloud selection

scheme can approach the global optimal assignment when the edge has sufficient compute resources and high inter-edge bandwidth, and (c) adding capacity to an existing edge network without increasing the inter-edge bandwidth contributes to network-wide congestion and can reduce system capacity.

In chapter 3, techniques to enable QoS in MEC are described using a concept of named-object abstraction applied to the virtual networks. A layer 3 virtual network framework is designed, implemented and evaluated which can provide the control mechanisms to connect distributed resources across network domains. Building on that, an application specific routing mechanism is provided to support requests based on cross-layer information information extracted from network and application. Moreover, the named-object abstraction is applied to achieve dynamic resource management using in-network resource slicing and devising mechanisms for the QoS control. Finally, the experimental results are obtained using a working prototype implemented usng the Click modular router software.

Chapter 4 provides a technique to handle dynamic load and mobility in MEC by migrating services across the MEC. Container migration is emerging as a potential solution that enables dynamic resource migration in virtualized networks and MEC systems. A traffic aware container migration approach is proposed, and validated with an end-to-end system implementation using a pure container hypervisor called LXD (Linux Container Hypervisor). The container migration model is then evaluated for real-time applications such as license plate recognition running in a mobile edge cloud scenario based on city-scale mobility traces from taxicabs in San Francisco. The system evaluation considers key metrics associated with application quality-of-experience (QoE) and network efficiency such as the average system response time and the migration cost for different combinations of load, compute resources, inter-edge cloud bandwidth, network and user latency. A specific compute resource and network-aware distributed resource migration algorithm called "ShareOn" is proposed and compared with alternative techniques using the San Francisco MEC model.

Finally, chapter 5 brings all the architectural components together by proposing a distributed control plane protocol and providing a detailed evaluation of techniques to

support low-latency in the MEC. A novel control plane protocol is under discussion to enable cooperative resource sharing in heterogeneous edge cloud scenarios to address the key design challenges including: (a) specification of a lightweight overlay control plane protocol for distributed edge, (b) evaluation of control protocol overhead and achievable system performance with cooperation between edge clouds, (c) specification of cooperative resource sharing algorithms (e.g., offloading, cluster computing and service chaining) enabled by the protocol, and (d) evaluation of end-to-end latency for specific real-time applications to be run on edge clouds. This evaluation is conducted via an experimental edge cloud setup on the ORBIT testbed at WINLAB.

Chapter 2

Mobile Edge Cloud Requirements

Edge clouds promise to meet the stringent latency requirements of emerging classes of real-time applications such as augmented reality (AR) [31] and virtual reality (VR) [32] by bringing compute, storage and networking resources closer to user devices [33, 34]. Edge compute resources which are strategically placed near the users in the access network do not incur the irreducible propagation delays associated with offloading of compute intensive tasks to a distant data center. In addition, the use of edge computing can also lower wide-area backhaul costs associated with carrying user data back and forth from the central cloud. AR and VR applications enable users to view and interact with virtual objects in real time, hence requiring fast end-to-end delivery of compute services such as image analytics and video rendering. Previous studies [35, 36, 37, 38] have shown that latency associated with AR or gaming applications can be reduced by migrating some of the delay-sensitive tasks computing tasks to local servers, while maintaining global state in the core cloud.

While edge clouds have significant potential for improved system-level performance, there are some important trade-offs between edge and core clouds that need to be considered. Specifically, core clouds implemented as large-scale data centers [5] have the important advantage of service aggregation from large numbers of users, thus making the traffic volume predictable. Further, service requests entering a large data center can be handled in a close to optimal manner via centralized routing and load balancing [39] algorithms. In contrast, edge clouds are intrinsically local and have a smaller scale and are thus subject to significantly larger fluctuations in offered traffic due to factors such as correlated events and user mobility. In addition, we note that edge computing systems by definition are distributed across multiple edge networks and hence are associated

with considerable heterogeneity in bandwidth and compute resources. Moreover, the data center model of centralized control of resources is not applicable to a distributed system [40, 41] implemented across multiple edge network domains, possibly involving a multiplicity of service providers.

A general technology solution for edge clouds will thus require suitable distributed control algorithms and associated control plane protocols necessary for realization. The unique nature of the distributed edge cloud system poses key design challenges such as specification of a control plane for distributed edge, distributed or centralized resource assignment strategies, traffic load balancing, orchestration of computing functions and related network routing of data, mobility management techniques and so on. MEC also has other advantages such as security to prevent data being sent away from the MEC. This thesis is focused on latency. In order to address these challenges, a simulation based system model is the foundation for understanding performance and evaluating alternative strategies for any of the above design issues.

This chapter presents an analysis of the scalability and performance of a general hybrid edge cloud system which supports latency-sensitive applications. The goal is to provide a better understanding of key system design parameters such as the proportion of resources in local cloud vs. data center, fronthaul and backhaul network bandwidth, relative latency/distance of core and edge clouds, and determine their impact on system level metrics such as average response time and service goodput. Using the model described here, we seek answers to the following questions: (a) How much load can an edge cloud network support without affecting the performance of an application; (b) How does the value of the application delay-constraint affect the capacity of the system; (c) What is the impact of offered load and resource distribution on goodput; (d) Under what circumstances can the core cloud perform better than an edge network and vice-versa; and (e) What is the impact of inter-edge (fronthaul) and edge-to-core (backhaul) network bandwidth on system capacity?

We use a simulation model to study a city-scale general multi-tier network as shown in Fig. 2.1 containing both edge and central cloud servers. The model is used to obtain system capacity and response time for an augmented reality application while analyzing

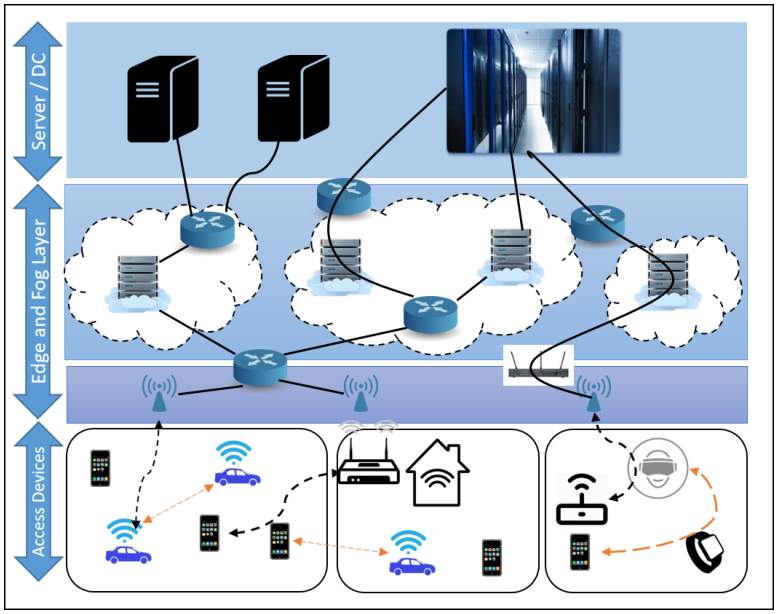

Figure 2.1: General Multi-tier Edge-cloud Network Architecture

the impact of key parameters such as resource distribution and fronthaul/backhaul bandwidth. A general optimization framework for the distributed system is proposed and compared with distributed algorithm approaches.

The rest of the chapter is organized as follows. Section 2.1 demonstrates the augmented reality application with two use-cases deployed at WINLAB to measure their computation (e.g., processing delay), networking (e.g., propagation and transmission latency) and storage (e.g., database use) requirements. The requirements demonstrate a need of edge cloud to achieve low-latency. Section 2.2 provides the simulation details including the system model with an emphasis on system design, and performance model to analyze edge clouds using a city-scale network including models for application, compute and latency. A baseline distributed resource allocation approach for selecting an edge cloud for an AR application is also detailed in Section 2.3. Section 2.4 presents the performance evaluation of the baseline approach. Section 2.5 proposes and evaluates a capacity enhancement heuristic (ECON) for real-time applications. Numerical results to compare ECON and the baseline are given in Section 2.6. Section 2.7 provides related work in the field and finally, Section 2.8 concludes the chapter.

2.1 Augmented Reality and Edge Clouds

Augmented Reality (AR) is gaining popularity in numerous fields such as healthcare, visualization, entertainment and education. Most of the commercially available AR devices like Atheer AiR [42], Microsoft Hololens [43] and Google Glass [44] have limited power, storage and on-chip computation capabilities; for example, currently the Hololens (first version) has ∼64 GB storage and ∼2GB RAM. Thus, these devices often rely upon offloading storage as well as compute to an architecturally centralized cloud server to ensure that the application response time requirements are met.

The Quality of Experience (QoE) perceived by a user running an AR application using cloud services is a complex combination of network bandwidth, network traffic and compute capabilities of the cloud. First, the bandwidth from end–user to a cloud data center is the minimum bandwidth available across all the hops in the network path, which could be significant when cloud is located far from the user. Second, the network traffic depends upon the network load and congestion, and varies for each individual local network. Edge cloud computing (denoted as "edge" in the following discussions) promises to alleviate the shortcomings of the cloud server by bringing computation, networking and storage closer to the user and providing fast response, context awareness and mobility support [45]. Therefore, edge computing can be viewed as having the same centralized cloud resources scattered at the mobile network edge and accessed through fast Wi-Fi or 5G access networks. This approach has the potential to provide tightly bounded service response time thereby creating a geographically distributed heterogeneous computing and communication system.

Edge computing complements the cloud infrastructure as edge clouds are resource limited in terms of bandwidth and compute. The multifaceted edge system therefore must be studied in conjunction with the existing core cloud for different user requirements, application types, edge assignments and QoS constraints. Thus, for a resource constrained system it is required to allocate resources per request while taking system capacity into consideration. This leads to a nonlinear optimization problem [46] due

to multiple factors affecting the capacity including but not limited to network bandwidth, resource availability and application type. In order to understand the capacity constraints of a hybrid edge cloud system for latency sensitive applications, we analyze the MEC using a set of sample AR application [47].

2.1.1 Use Case Scenario

Figure 2.2(a) shows the process flow of our implementation of demo AR applications using Microsoft Hololens. A client sends a continuous video stream to the edge server which processes the information based upon application type and returns output to the client. The video stream (30 fps) is processed by OpenCV [48] 3.3 running on Intel i7 CPU 980, 3.33GHz and 15GB RAM. The edge server is connected to the client in two hops: (i) edge to first hop router (bandwidth: 932 Mbps) and (ii) router to Hololens (bandwidth: 54 Mbps). The following use-cases are evaluated.

Smart Navigation. A user enters a building. The edge in the building has her contextual information from the calendar entries and GPS. As shown in Fig. 2.2(b) the user is navigated to meet a person in the building using a set of cubes appearing on the device as she moves. Achievable latency is critical here because real-time activities of the user can be disrupted by late arrival of AR information.

Annotation based assistance. In this scenario, a user looks at an object having a set marker through Hololens with an intention to get supplementary information about the object. In Fig. 2.2(c), user looks at the printer and its status, ink level, number and current jobs are annotated on the user's display.

2.1.2 Application Flow Timing Diagram

Figures 2.3(a) and (b) show (not to the scale) the timing diagrams of a sample packet flow in the system for smart meeting and annotation-based assistance applications respectively. The network delay in both the cases is kept below 10 ms by deploying edge cloud services a single hop away from the AP. In both the scenarios, the processing delay, path finding in the navigation and OpenCV image processing in the annotation

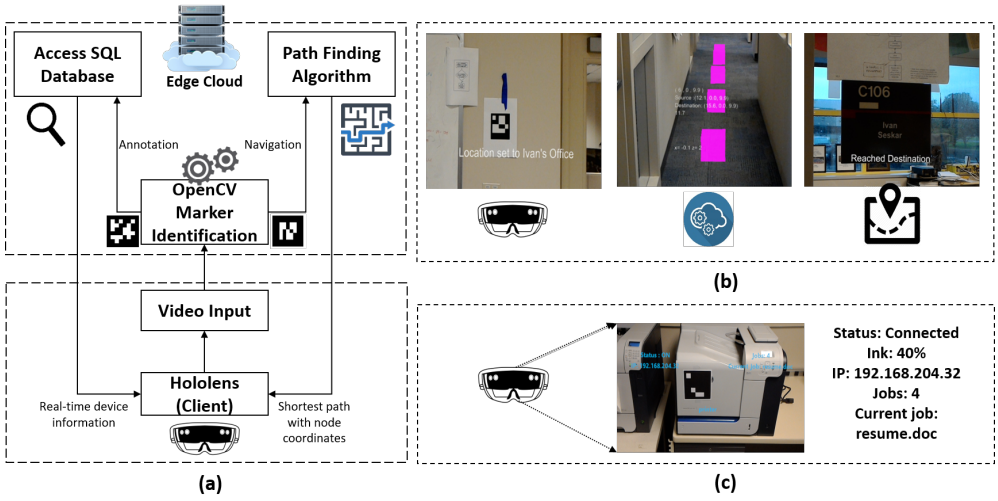

Figure 2.2: AR Use-case Scenario Set-up: (a) AR Application Flow (b) Smart Meeting Application using Indoor Navigation and (c) Annotation based Assistance

application, can be a major bottleneck. The following techniques are used in our implementation to lower the total response time as compared to the traditional core cloud based services: (i) reduction of network latency via higher bandwidth and closer edge cloud service; (ii) passing minimum processed information to the client such as end-to-end coordinates (8 Bytes) per query in case of the navigation and 64-1500 Bytes per frame processed for the annotation application, and (iii) offloading multiple tasks to the edge cloud to minimize local processing at the UE. The AR implementation serves as a guide to the parameters used in the system model described in the next section, which assumes a low-latency requirement to run AR applications with an acceptable subjective quality [38]. Using our deployed AR applications, this section confirms that: (a) the total application latency can be brought down by reducing the number of hops and increasing available access bandwidth, and (b) although edge cloud lowers the network latency, application processing latency contributes significantly to the total latency for AR applications.

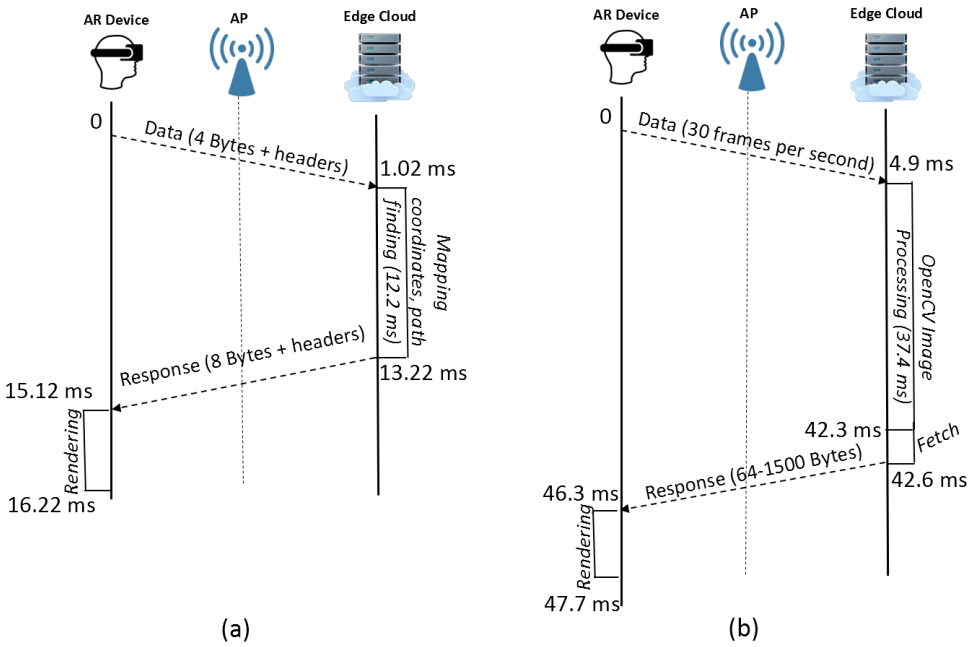

Figure 2.3: Timing Diagram for the AR Applications: (a) Smart Meeting Application using Indoor Navigation and (b) Annotation based Assistance.

2.2 System Model

2.2.1 System Design

The system diagram of the hybrid edge cloud under consideration is shown in Fig. 2.4. Each AP is equipped with an edge cloud with a configurable compute resource capacity. In general, a compute resource represents a machine or a group of machines (cluster) also known as cloud or edge rack. A rack has limited capacity to support users for their computational requirements. For instance, an AR application requires computation to process video/image stream and receive their response back from the server. The edge rack in our design has maximum five processors each having 3.33 GIPS processing speed. The central cloud server is placed at Salem, Oregon (OR; location chosen to relate with commercially available central clouds) which again has a configurable capacity. The compute capacity is defined as the number of servers available at the edge cloud and/or at the central cloud. The inter-edge bandwidth is varied from 1 Gbps to 100 Gbps and AP-Cloud bandwidth from 10 Gbps to 500 Gbps. The special case of unconstrained

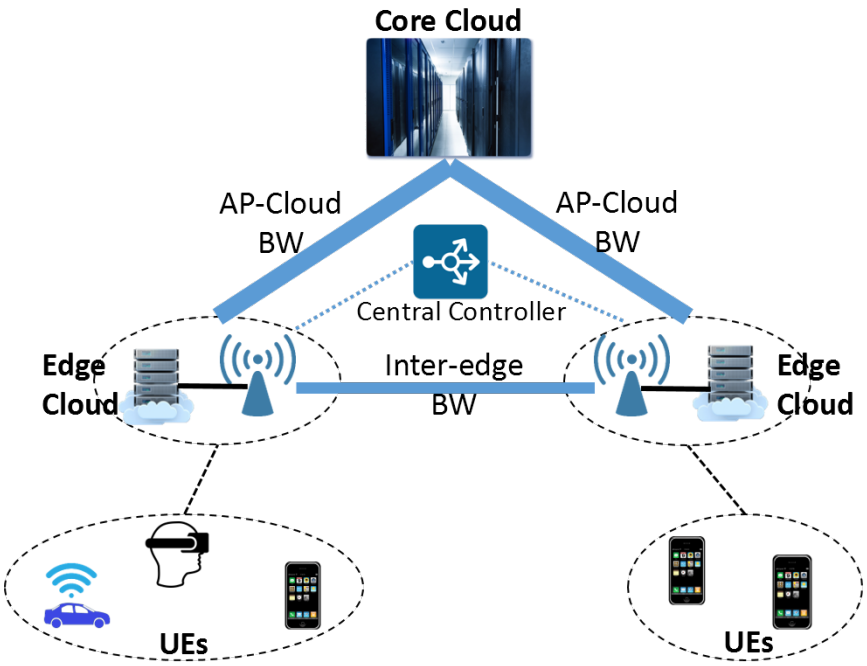

Figure 2.4: Hybrid Edge Cloud System Diagram

inter-edge and AP-cloud bandwidth is also considered. The central controller has the capability to collect network and compute parameters from all the edge clouds and the core cloud. The system design parameters are listed in Table 2.1.

In this study, the total amount of compute available at the edge clouds and core cloud is assumed to be fixed. This assumption holding the compute cost constant allows us to fairly analyze the impact of varying other key system parameters such as % of edge servers or core/edge bandwidth. In our simulation, we increase the resource density of already deployed edge clouds by removing and redistributing compute resources from the central cloud thereby keeping the overall compute resources for the whole system unchanged.

We use Chicago, the third most populous city in US, as a test-case considering locations of 11,00 WiFi APs [49] as shown in Fig. 2.5. The number of hops from Chicago to OR varies from 10 to 20 (including switches) and takes around 5-6 hops to reach the cloud server gateway whereas the average latency in US ranges from 13 ms to 106 ms [50] based on a ping test of 64 bytes packet from various locations. It

Table 2.1: System Design Parameters

Parameter	Value/Range
AP-Cloud Bandwidth	10–500 Gbps
Inter-edge Bandwidth	1-100 Gbps
Core Cloud Resources	0, 20, 40, 60 or 100%
Edge Cloud Resources	0, 20, 40, 60 or 100%
Core Cluster	0-5500 servers
Edge Clusters	0-5500 servers
AR Latency Requirements	50-100 ms

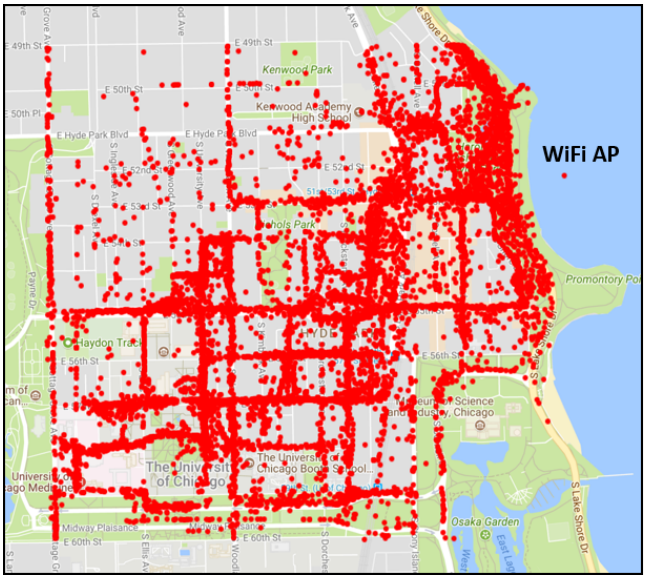

Figure 2.5: Wi-Fi APs Placement in Chicago City

is to be noted that the AR application's bit rate increases rapidly with resolution for instance a 160x120 pixels video needs around 1.7 Mbps whereas a 640x480 pixels video requires 27 Mbps continuous uplink bandwidth (assuming 30 fps, 24 bit per pixel), and 432 Mbps for 1920x1080 video. The response from the server is sent to the UE in the packets of size between 100 to 1500 Bytes per processed frame. Note that the uplink bandwidth requirement for an AR application is more than the download bandwidth due to its uplink video/downlink processed information characteristic which is quite different from most web traffic today. We model the network based on the type of application and its latency requirement.

We run an AR application at the UE which sends a video stream to the server while server computes the contextual information and sends back the output to the user. The application is annotation-based assistance using AR wherein a user gets information about surrounding annotated on his AR device as described in Section II. Annotation-based assistance can be used in various application scenarios. For example, a policeman looks at a license plate of a car while driving and the information about the owner gets displayed on the device. The license plate can also be run against a list of stolen car and can be immediately reported to the policeman.

Table 2.2: Simulation Parameters

Parameter	**Value**
Area	5.18 km^2
Number of APs	1.1K
Maximum Number of Users	55K
Distribution of Users	Random
Bandwidth (Uplink)	27, 150 and 300 Mbps
Bandwidth (downlink)	54, 300 and 600 Mbps
Packet Size	1500 Bytes
Edge Resources (baseline)	5 Machines
α	0.1
β	1
w	0.5
p	10

2.2.2 Performance Model

In this section, we describe system modeling aimed at evaluating user performance and system capacity as a function of key design parameters. A multi-tier edge-cloud system as shown in Fig. 2.4 can be divided into user, network (data and control) and computation plane. Our system design is a hierarchical composition of compute and network elements. The computation at edge or cloud is similar in functionality but different in terms of resources availability as the core cloud has a single big pool of shared resources while each edge cloud has limited resources closer to the user. The following discussion presents application, compute and latency modeling.

Application

In our model, the application is defined by a four tuple $< V, G, S, L >$ where V denotes the computational task per unit time ranging from $[1, n], n \in Z^+$. Each AR application requires these tasks to be completed within a specified real-time threshold latency in order to be useful to the AR application. In case a task is not completed within the application latency threshold, the goodput of system goes down. G denotes the geolocation of the UE. A city is considered to be a collection of G_i blocks (assume as cells of a cellular network), $i \in [1, N]$ where N is the total number of geographical blocks. For simplicity, we divide the geographical area into square G_i's. Analyzing the users served by each block provides us meaningful information if we need to upgrade the capacity of an edge cloud in the block if it is available. Binary $S \in \{0, 1\}$ denotes the availability of the edge cloud in the geographical area G of a user. Unavailability of an edge cloud may mean that there is no physical edge cloud present or the edge cloud of that region has run out of capacity in which case, a neighboring edge cloud can be chosen or the user can be routed to the central cloud. For delay-tolerant applications, routing a user to the central cloud frees resources at the edge to serve latency-sensitive applications. Finally, $L \in (0, d_{\max})$ represents the maximum tolerable latency for the said application.

Compute

The delay due to computation is modeled using a multi-server queuing model. The edge cloud is like a mini data center where tasks arrive from geographically distributed users, processed by the available resources in the edge cloud and depart. Therefore, as the number of transactions in the system increase when the system load rises these tasks are queued till they are processed. This scenario can be best represented by employing an M/M/C queuing model[51]. Each edge or central cloud processes multiple service requests in a work-conserving FCFS queue with assumed infinite buffers. The overall latency is dependent on the arrival rate λ, service rate μ and the number of servers c. It can be noted that as the system computation power is constant, increasing capacity

at the edge will mean removing equivalent resources from the central cloud implying a rise in queuing delay at the cloud.

For a given set of static users, the system load is proportional to the number of active users and the rate of application requests per second. In our model, Load=0.1 is defined as 10% of the total users are running the AR application. In general, average time spent by a task in the server is the sum of transmission delay, queuing delay and processing delay, which is calculated using the M/M/c queuing model [51] as given by Eq. 2.1–2.3.

$$d_{node} = W + \frac{1}{\mu} + t_{tx} = P_Q * \frac{\rho}{\lambda(1-\rho)} + \frac{1}{\mu} + t_{tx} \tag{2.1}$$

$$P_Q = \frac{(c\rho)^c}{c!}\frac{1}{1-\rho}p_0 \tag{2.2}$$

$$p_0 = \left[\sum_{k=0}^{c}\frac{(c\rho)^k}{k!} + \frac{(c\rho)^c}{c!}\frac{1}{1-\rho}\right]^{-1} \tag{2.3}$$

Here, d_{node} is the total time spent by a task V at the edge cloud or the core cloud, W is the wait time in the queue, P_Q is the queuing probability, ρ is the server utilization, c are number of servers at each edge or total server at the cloud, and p_0 is the initial probability. In view of shared bandwidth on inter-edge links, the transmission time t_{tx} can be simplified as b_{link}/r_{users} where b_{link} is the total bandwidth of a link and r_{users} are number of total tasks run by all the users at an edge. For large c, to avoid involved calculations in Eq. (2), we split cloud computing resources into set of uniform clusters where a selected cluster is one serving the lowest number of concurrent tasks.

Latency

The overall latency of an application has several components including irreducible propagation delay, the transmission delay, routing node delays and the cloud processing time. For a core cloud server, which carries aggregated traffic, there is also a Software Defined Networking (SDN) switching latency. As the number of users increase in a geographical region, the bandwidth is shared among them costing more transmission delay. For a

cloud only model when there are no edge servers, the total cloud latency can be stated as:

$$L_{cloud} = \alpha * D_{\min(UE,APs)} + \beta * D_{AP-cloud} + d_{node} \tag{2.4}$$

Eq. 2.4 shows that a closest AP is chosen to route a user to the cloud. Here, α and β are the proportionality constants to account for the propagation delays, $D_{\min(UE,APs)}$ is distance from UE to nearest AP and $D_{AP-cloud}$ is the distance from AP to the central cloud. When resources are available at the edge, the total edge latency can be represented as:

$$L_{edge} = \alpha * D_{\min(UE,APs)} + d_{node} + d_s \tag{2.5}$$

In Eq. 2.5, $d_s \geq 0$ is the control plane switching latency from an edge at AP to another AP's edge in case of unavailable resources which is assumed to be between 1–5ms. The response time for an application is the sum of transmission delay, propagation delay, switching delay (if any), queuing delay and computation delays in both the cases. The specific values of α and β are selected to relate with the distance between users and cloud components.

A core cloud-only system is defined as one with no edge cloud available. The edge-only system does not have any core cloud and if the load exceeds the available computational resources, a request is queued until it is processed. We also consider hybrids of core and edge based on the percentage parameter that splits computing resources between the two.

2.2.3 Edge Selection for an AR Application

Edge selection in a system for a given traffic load can be achieved using multiple approaches. The network routing information that is available to all the routers can be used to deliver the service request to the nearest edge cloud — the edge cloud then independently decide to serve the request based upon resource availability or can route the user to the central cloud. A queuing model is used to estimate the service time for a

Algorithm 1: Finding neighboring edge with available resources for an AR application

```
1 function AvailableNeighbor (a, b);
  Input  : Neighbor resource and bandwidth s_i and b_i
  Output: TorF
2 Condition: TotalDelay_Edge ≥ delay_th
3 while(NeighborEdge)
4 if TotalDelay_NeighborEdge_i ≤ delay_th then
5 |   return TRUE;
6 else
7 |   return FALSE;
8 end
```

request apart from networking delays (control plane), propagation delays and transmission delays (available bandwidth). The nearest edge approach works well for scenarios with evenly distributed users and network resources. However, this simple nearest edge cloud routing strategy does not work well when the user distribution is not geographically uniform ascertained by our simulation showing only 10% improvement in the average system response time as compared to a cloud-only system.

An alternative distributed approach improves upon simple anycast by having routers maintain compute resource availability states of neighboring edge clouds. This may involve the use of overlay protocols to exchange cloud state in a distributed manner [52, 53]. A user is routed to the nearest edge first which makes one of the following decisions: (i) serve the request, (ii) route to a neighboring edge with available resources, or (iii) route to the central cloud. The decision at the edge is based upon application requirement and traffic load. For an AR application, the decision metric selects the closest edge to the UE which can serve the UE in $L_{edge} \leq d_{\max}$.

2.2.4 Baseline Approach

Algorithm 1 shows the pseudo-code for the baseline edge cloud selection approach adopted in our study. The algorithm is invoked whenever the default edge cloud is unable to serve the user's demand (line: 2). It then scans the states of neighboring edges to find the best edge which can serve the user within the specified latency threshold. This approach relies upon shared resource and bandwidth information among

neighbors. The list of neighbors is defined as p closest edge clouds from the current edge location. For finite p the order of state update messages to be exchanged is $\sim N*p^2$ where N is the number of edge clouds, and is thus an acceptable overhead for small to moderate values of p.

This section detailed our system and performance model. A baseline algorithm which scans the states of neighboring edge clouds to find the best edge which can serve the user within the specified latency threshold is developed. Next section evaluates the performance of baseline algorithm.

2.3 Performance Evaluation of Baseline System

In this section we discuss the capacity of different edge cloud systems with respect to traffic load, resource distribution and inter-edge bandwidth. Consider a system with following compute resources: (i) core cloud only, (ii) edge cloud only, and (iii) core cloud plus edge cloud, where in each case, the total amount of resources are same. Major system parameters used in the simulation are summarized in Table 4.2.

2.3.1 Impact of Network Bandwidth Parameters

Figure 2.6 illustrates the impact of constraint bandwidth AP-cloud system on the average response time. Here, the total bandwidth limit is set between edge network and the core cloud cluster. For a 500 Gbps AP-cloud bandwidth, for given system, the average response time compares with that of an unconstrained bandwidth case while for 50 Gbps case, it rises exponentially as the load increases. In case of lower bandwidth cases like 10 Gbps and 25 Gbps, the system is unable to handle higher load. As a bandwidth-constrained cloud system cannot compete with an edge-only system in terms of response time, further discussions in this work will assume a bandwidth-unconstrained cloud.

Figure 2.7(a) plots the average response time for the core cloud as well as edge only system with different inter-edge bandwidth. On one hand, the extreme fronthaul bandwidth of 100 Gbps edge-only compares with the unconstrained bandwidth edge-only system and therefore all the edge resources are utilized. On the other hand, after

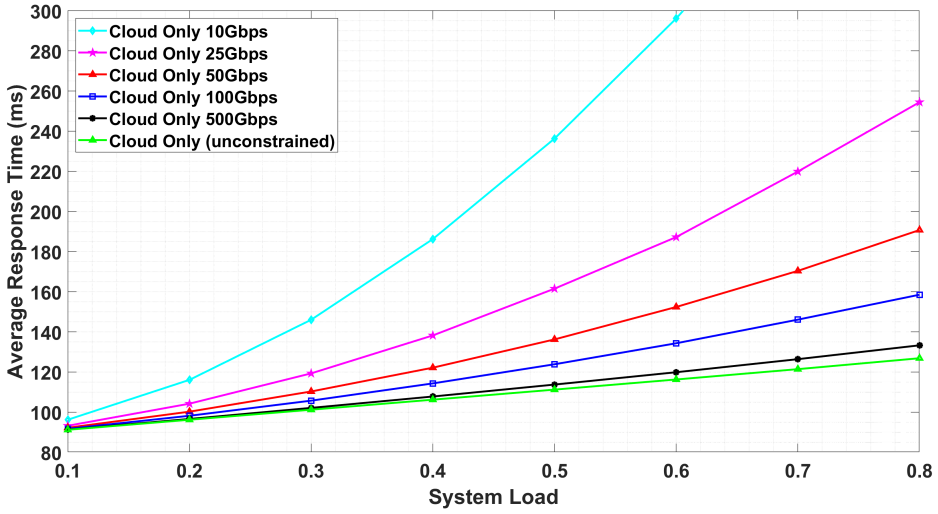

Figure 2.6: AR Application Average Response Time for Core Cloud only System with Increasing System Load and Different Uplink Bandwidth

the system fills up at Load=0.7, core cloud only system outperforms the edge only system with 1 Gbps inter-edge bandwidth. The reason is that for the baseline case, when an edge fills up the capacity, it routes the request to a neighboring edge utilizing inter-edge bandwidth. As the finite inter-edge bandwidth is split between multiple application flows, the propagation delay and queuing delay rise which in turn increases the average response time for higher load. In the baseline approach, the edge decides whether to send the request to a neighboring edge or to the central cloud. For 1 Gbps inter-edge bandwidth case, the average response time for Load=0.1 is as low as 30 ms while for Load=0.8 case, it rises to 130 ms as the bandwidth exhausts and queuing delay rises. A delay more than 100 ms is unsuitable for most of the AR applications. As the bandwidth doubles, for Load=0.8 case, the average response time is $\sim$95 ms. Increasing bandwidth lowers the average response time for a completely loaded system but beyond 10 Gbps there is no significant advantage visible for the baseline case as there are still significant queuing delays for a loaded edge at an AP (or neighboring AP). After a load point, there is no dip in response time irrespective of how good the fronthaul connectivity between edge clouds is. In this case, there is a crossover around Load=0.7 so we compare the CDF of core cloud only and edge-only with the 1 Gbps

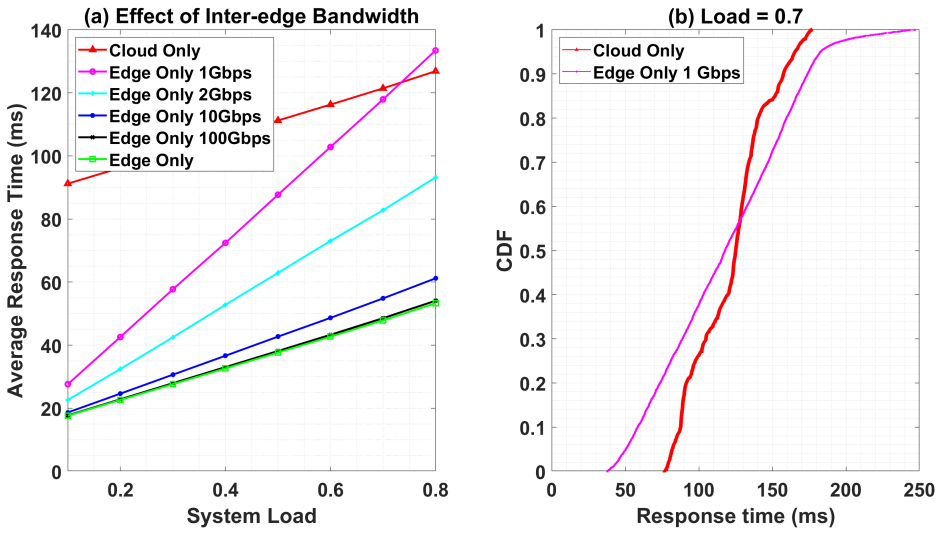

Figure 2.7: Average Response Time Comparison for Core Cloud and Edge Only System, with Different Load and Inter-edge Bandwidth for Baseline

case in Fig. 2.7(b). A linear rise in response time can be observed for the static load case implying that the inter-edge bandwidth of 1 Gbps is insufficient to run such a heavily loaded system.

2.3.2 Impact of Resource Distribution

In this section, we analyze the impact of the compute resource distribution between the core cloud and edge cloud on the average response time. There are a total of 5.5K processors each having 3.33 GIPS speed available as compute resources which are equivalent to 1.1K full edge racks. The compute resources are distributed between edge and cloud. In the simulation model, CE82 implies that 80% compute resources are available at the cloud and 20% are placed at the edge near the APs and so on.

Figures 2.8(a) and (b) compare average response time in CE28 and CE82 for the baseline with respect to inter-edge bandwidth and load respectively. Response times for inter-edge bandwidth of 10, 50 and 100 Gbps are close to each other for all the load cases for both scenarios. This implies that increasing inter-edge bandwidth indefinitely cannot improve the system performance when using the simple scheme of filling neighboring edge resources. Figure 2.8(a) also highlights the fact that when edge resources

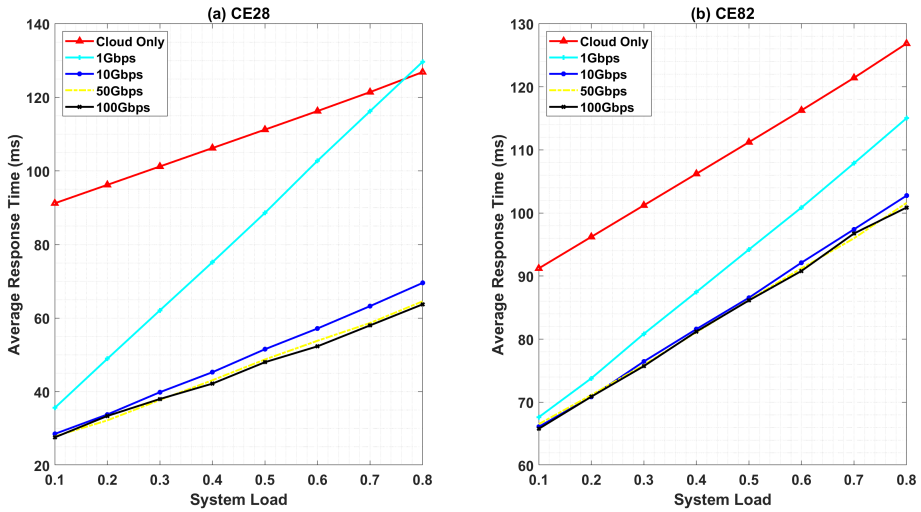

Figure 2.8: Average Response Time for Edge Cloud System for Different Load, Resource Distribution and Inter-edge Bandwidth for Baseline

are higher than the core cloud for a low inter-edge bandwidth, beyond a load point, the core cloud-only system performs better. This means that for a highly loaded system, if fast edge connectivity is unavailable, it is better to use the core cloud. Here, cloud only means that there are no edge clouds in the network and APs are connected to the cloud with unconstrained bandwidth.

2.3.3 Impact of AR Application Traffic Parameters

Figure 2.9 establishes that a limited inter-edge bandwidth makes system very susceptible to load. For the CE28 case, when the cloud-edge resource distribution is 20%-80% and inter-edge bandwidth is 1 Gbps, average response time increases at a faster rate than that of the CE82 case. The reason is that in the baseline scenario for CE28, an edge might be able to find a neighbor with available capacity but the connectivity is not sufficient to reach to that neighbor. In the case of lower or no edge resources, the core cloud is immediately favored and therefore performs better than the edge cloud scenario as can be observed from the crossover point at Load>0.7.

One more point of interest in Fig. 2.9 is between Load=0.5 and Load=0.6 where all the CE cases intersect. Figure 2.10 shows the average response time with different

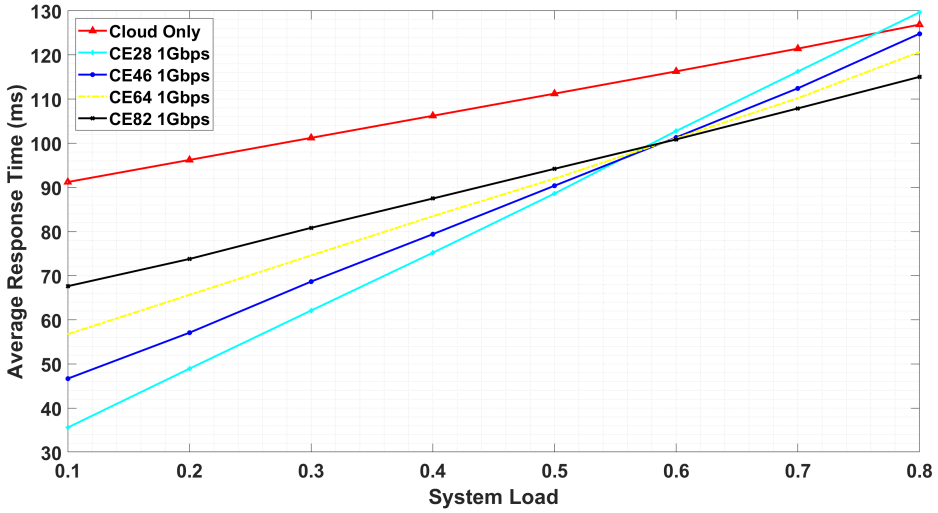

Figure 2.9: Average Response Time for Edge Cloud System with Different Load and Resource Distribution for Baseline. Inter-edge Bandwidth=1Gbps.

inter-edge bandwidth and resource distribution for baseline when Load=0.5. Here, for the CE82 case, increasing inter-edge bandwidth does not boost the system performance as compared to the CE28 case because for the low edge resources case, increasing inter-edge bandwidth cannot decrease the processing delays at the edge. For a system with high edge resources, a higher inter-edge bandwidth is therefore needed to maintain AR performance.

Similarly, for the Load=0.6 case, Fig. 2.11 plots average response time vs. resource distribution for different inter-edge bandwidths. Again, for a 50 Gbps inter-edge bandwidth system, a faster drop in the average response time can be observed for the CE28 case when 80% resources are at the edge. For a 1 Gbps inter-edge bandwidth system, the average response time is slightly higher for the CE28 system than for the CE46 system.

Using our designed system and performance model, we make following observations for the baseline scenario: (a) for unconstrained compute resources, the edge cloud continues to perform better than the core cloud due to its vicinity to the users (lower network latency), (b) increasing core network bandwidth beyond a load point does not lower the total application latency as the compute latency takes over, (c) for higher

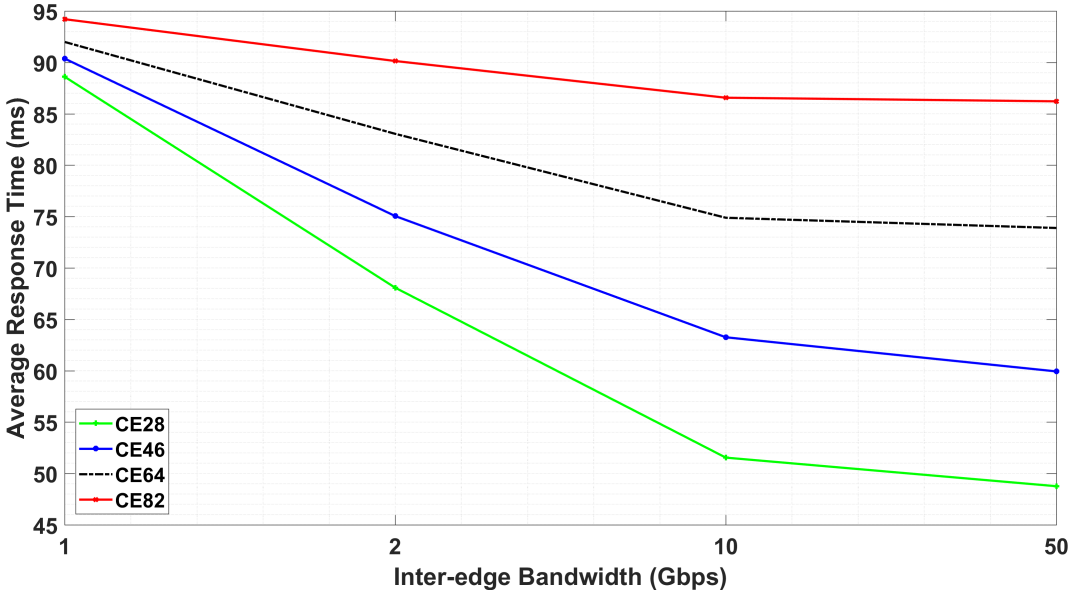

Figure 2.10: Average Response Time for Edge Cloud System with Different Resource Distribution and Inter-edge Bandwidth for Baseline. Load=0.5.

system load, the propagation delay and queuing delay rise because finite inter-edge bandwidth is divided among multiple application flows, (d) indefinitely increasing fronthaul edge cloud connectivity does not improve the response time after a load level, and (e) for lower inter-edge bandwidth case, distributing more resources at the edge clouds only worsens the application performance.

2.4 ECON: Enhanced Capacity Edge Cloud Network

The baseline approach considered in the previous section relies on a distributed control to select the best available neighboring edge cloud which might be sub-optimal in terms of overall system capacity. A more general approach is to select an edge cloud based upon global information about network and compute resources available at a logically centralized point such as an SDN controller. The idea is to use the complete network view before assigning an application/user to an edge cloud or deciding to route it to the core cloud. We call this approach Enhanced Capacity Edge Cloud Network (ECON). This section describes the ECON method and compares its performance with the baseline method.

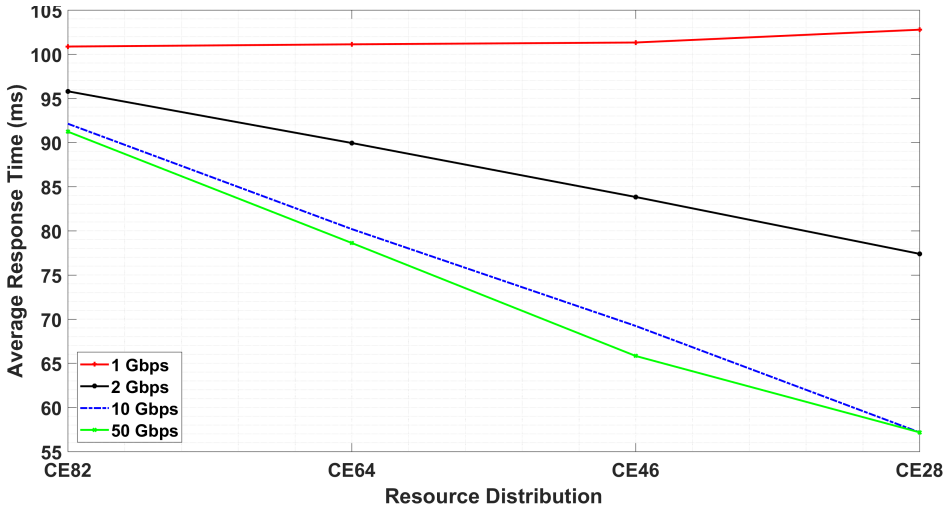

Figure 2.11: Average Response Time for Edge Cloud System with Different Resource Distribution and Inter-edge Bandwidth for Baseline. Load=0.6.

Definition 1: An edge or cloud is *"usable"* for a request i if the latency L_i^a for the user running an application a is below the latency threshold for given application L_{Th}^a i.e. $L_i^a \leq L_{Th}^a$. Here, L_i^a is simply equal to L_{cloud} or L_{edge} with different d_{node} and d_s.

A *"usable"* server is best for a user request in terms of service quality whereas the overall system capacity might not be optimal with this assignment. For example consider a user's application latency threshold 110 ms which may be assigned to an edge server serving request within 30 ms. This assignment will hamper performance of another needy user who required 35 ms latency but cannot be accommodated due to unavailable resources at the edge.

Definition 2: *"delay-constraint (%)"* of an edge-cloud system is defined as the number of requests out of hundred served below the application response time threshold, L_{Th}^a. For a specific value of L_{Th}^a, the delay-constraint can also be interpreted as system capacity. For instance, a delay-constraint of 10% for a 15 ms threshold implies that system can accommodate only 10% of the total requests and 90% requests will only consume resources to lower the goodput. This means for 90% of the requests, the assigned edge resources are *"not usable"*.

Percentage delay-constraint, $C = (n_{Th}/N) * 100$, where n_{Th} are requests served

within threshold response time and N are the total number of requests in the system. A system with high C for a threshold is required to run latency sensitive applications.

2.4.1 "Usable" Edge-Cloud Optimization

Assigning requests to a *"usable"* server is similar to capacity optimization of an edge-cloud system for given compute as well network resources and application delay fulfillment. This problem is equivalent to the maximum cardinal bin packing and hence is NP-hard [17, 54]. We can model the global optimization to maximize usable server s for N requests, where each request i is assigned to the server s, as:

$$\max_{s} \sum_{n \in N} I_{\{s_n > 0\}} \tag{2.6}$$

subject to:

$$L_i^a(s) \leq L_{Th}^a, \forall s_n > 0, n \in N \tag{2.7}$$

$I_{\{s_n>0\}}$ being the indicator function with values 1 or 0 depending upon if such a server is available or not for a given request which means if it can serve the request in application response time threshold. Mapping users to *"usable"* server is NP-hard problem as explained earlier thus requiring an alternative approach.

The total average processing delay, d_{comp}, at the cloud or edge, comprise of a waiting delay in the queue and a processing delay associated to the type of application. At each node, there is a transmission time, t_{tx} associated with each task V, adding which to d_{comp} provides total time, d_{node}, spent at a server. Therefore, for such a system, we can formulate Eq. (2.6) as minimizing d_{node} of the system for all the users, while compromising on the optimality, instead of a *"usable"* server problem as follows:

$$P_1 : \min \sum_{i=1}^{M} (\sum_{j=1}^{N} d_{proc}^{j,i} + d_{tx}^{i} + d_{s}^{i}) \tag{2.8}$$

subject to:

$$L_j^a \leq L_{Th}^a, \forall j \in N \tag{2.9}$$

$$b_{\min}^{i,up} \leq b_i^a \leq b_{\max}^{i,up}, \forall i \in M \tag{2.10}$$

$$b_{\min}^{i,down} \leq b_i^a \leq b_{\max}^{i,down}, \forall i \in M \tag{2.11}$$

$$\sum_{i=1}^{M} c_i \leq C \tag{2.12}$$

Equation 2.8 defines the optimization problem with Eq. (2.9) as delay constraint, Eq. (2.10) and Eq. (2.11) as bandwidth constraints for uplink and downlink each user application and, Eq. (2.12) as capacity constraint of each node respectively. As explained earlier, b_i^a can be computed as b_i/r_{edge}. Again, the problem is similar to maximum cardinality bin packing problem and is NP-hard. Therefore, to find the *"usable"* server, we need to fix a user to a nearby edge and find the Pareto optimal edge for the next user sequentially satisfying the application latency constraint. This can be done by omitting the switching delay. Therefore, the problem can be simplified as (with same constraints as above) follows assuming d_{tx}^i constraint is satisfied by bandwidth splitting for each request.

$$P_2 : \min \sum_{i=1}^{M} \sum_{j=1}^{N} d_{proc}^{j,i} \tag{2.13}$$

Equation 2.13 establishes that for a latency sensitive AR application, finding the *"usable"* server for a user means we need to place the task to a server which is *nearby* to the user in strict network sense having low load, latency and high available bandwidth. The delay minimization objective function fills up the edge resources before routing a task to the central cloud. The latency and bandwidth of chosen server are estimated using the exponential moving average: $x_p * w_x + (1 - w_x)\overline{x}$, with w_x as weight factor for x, x_p is the previous value, $\overline{x}$ is the previous average and x is latency or bandwidth parameter. We call this approach ECON and results are compared with the baseline in next section.

2.5 ECON vs. Baseline

2.5.1 Resource Distribution and Inter-edge Bandwidth

ECON relies upon filling up the edge resources before routing to the central cloud. Figures 2.12(a) and (b) compare average response time for CE28 and CE82 cases when the inter-edge bandwidth is 1 Gbps. For an edge-favored CE28 scenario in Fig. 2.12(a), ECON and baseline have similar performance because finding an available resource in

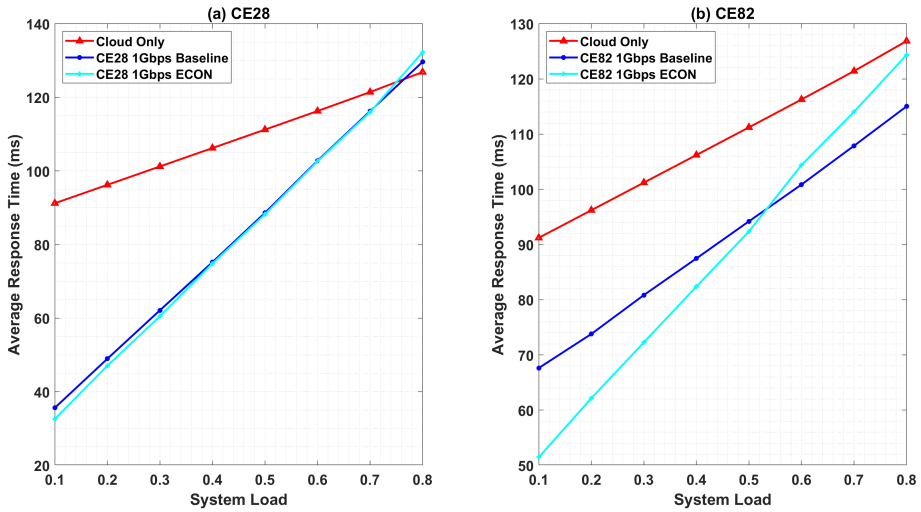

Figure 2.12: Average Response Time Comparison for ECON and Baseline, for Different Load and 1 Gbps Inter-edge Bandwidth

ECON is equivalent to finding a neighbor in the baseline which has high probability when edge resources are 80%. When the resources are cloud-favored i.e. CE82 in Fig. 2.12(b), for a lightly loaded system, ECON performs better as it is able to find the resources anywhere in the network without additional queuing delays at the edge. For a highly loaded system, finding an available edge is more expensive than routing the request to the cloud itself and therefore baseline outperforms ECON in case Load>0.5.

2.5.2 Application Delay Constraints

Figure 2.13 presents the delay-constraints for unconstrained fronthaul bandwidth for an edge-cloud system for the CE82 case when Load=0.1. As application latency threshold increases, delay-constraint rises meaning if an application has a latency threshold of 100ms, about 60% requests can be fulfilled by the cloud-only system whereas the edge-only system will be able to fulfill all the requests. As shown in the plot, without inter-edge bandwidth limits, ECON performs better than the baseline as it fills up maximum edge resources before routing any request to central cloud.

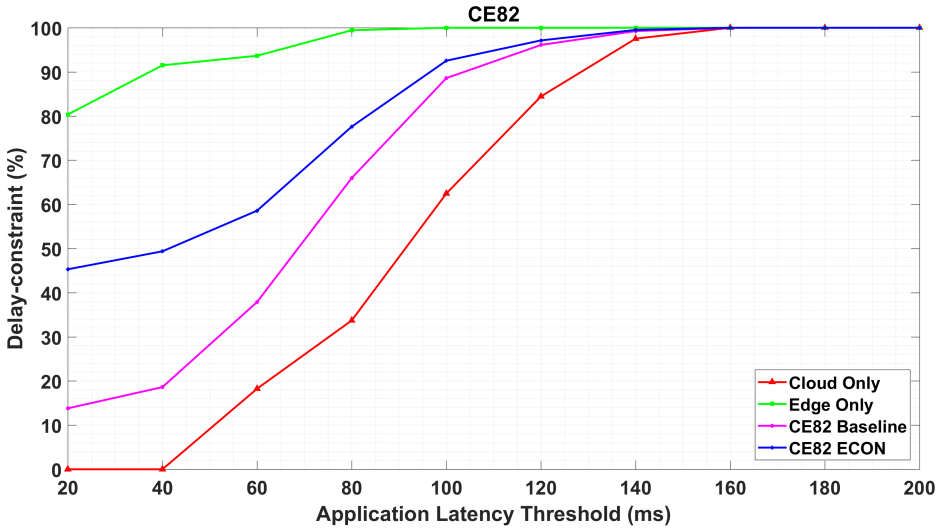

Figure 2.13: Impact of Application Latency Threshold on Delay-constraint Percentage for ECON and Baseline without Inter-edge Bandwidth Constraints

Edge-favored vs. Cloud-favored

Figures 2.14(a) and (b) compare edge and cloud favored resources respectively when inter-edge bandwidth is 10 Gbps. Figure 2.14(a) shows that for an edge-favored case when most of the resources are available at the edge, a baseline neighbor selection scheme performs equally well as ECON which selects the best of all edge resources for the request. For the cloud favored resource case shown in Fig. 2.14(b), ECON performs better than baseline as each of the edges has sufficient bandwidth to reach a far away available edge resource. Therefore, when sufficient bandwidth is available, it is better to choose an edge even if there are fewer resources available as the queuing time at an edge can be compensated by faster request transfers. On the other hand, if the inter-edge bandwidth is low, instead of trying to maximize edge resource utilization, it is good to send the request to the cloud if the application can withstand the resulting delay.

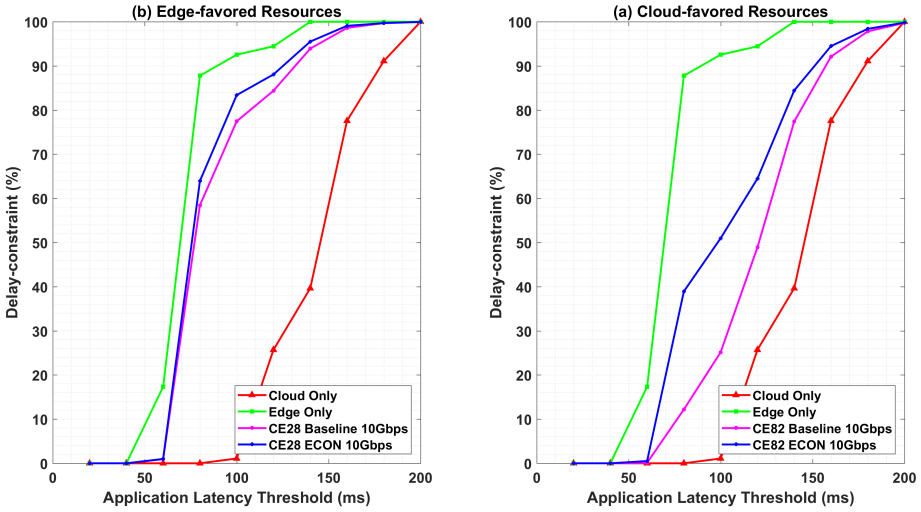

Figure 2.14: ECON and Baseline Comparison for Edge and Cloud Favored Resources (Inter-edge BW=10 Gbps)

Goodput

As discussed earlier, AR applications are delay sensitive and discard packets which arrive late. Goodput is defined as the number of useful (on time) bits per second delivered to UEs running the AR application. Therefore, even when the system throughput is high, the goodput could remain low due to high proportion of late arrivals. The capacity improvement can be studied by analyzing a geographic block, $G_i's$ level of goodput using our simulation tool. If goodput is lowest in a block, it indicates a need to augment additional edge resources to the serving edge. Figure 2.15 shows the normalized ratio of goodput of ECON and goodput of baseline for different resource distribution and load. For an unconstrained inter-edge bandwidth system, the goodput ratio of a cloud-favored system is more than that of an edge-favored one as ECON tries to find the best available edge resource as compared to the neighbor selection baseline scheme. In a cloud-favored system, the edge has minimal resources and therefore each edge requires sufficient bandwidth to transfer requests to other edges which may be far away. The edge-favored system cannot be significantly improved with ECON as there are ample neighboring edges available from the baseline and therefore finding a more optimal edge

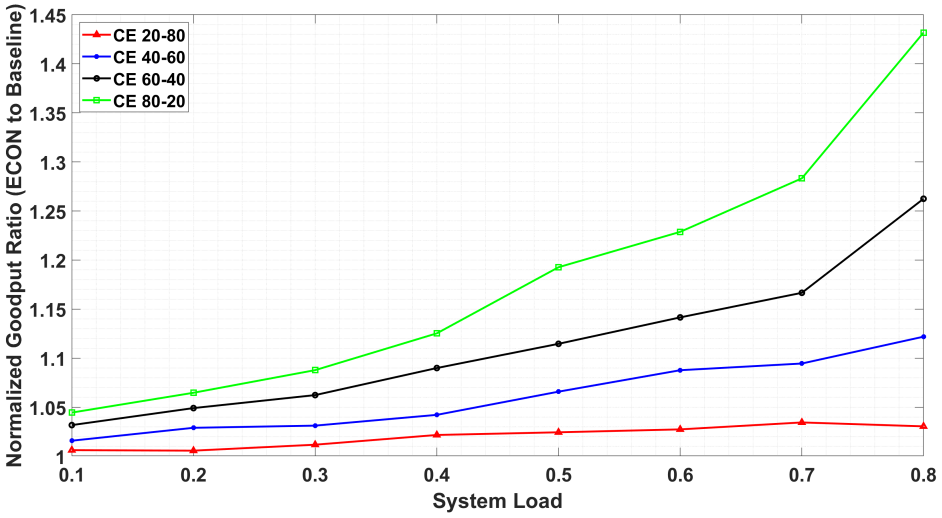

Figure 2.15: Impact of Load on Goodput Ratio of ECON and Baseline in an Edge Cloud System for Real-time Applications

tends to increase the network delay. Also, as the system load increases, there is a rise in the queuing delay at the edge server and therefore the system performance is similar for ECON as well as baseline in this case.

This section compared baseline scenario with a global edge assignment approach called ECON. We found that: (a) for an edge-favored resource system, ECON and baseline have similar application response time performance, (b) for a cloud-favored resources and lightly loaded system, ECON performs better than the baseline, (c) maximizing edge cloud's usage for lower inter-edge bandwidth hampers the average system response time, and (d) for elastic applications such as email, a cloud-only system is sufficient and can even perform better as compared to an edge-cloud system with low bandwidth.

2.6 Related Work

Edge cloud solutions have been proposed for a number of emerging scenarios including Internet of Things (IoT) [55], Cloud of Things (CoT) [56, 57, 58, 59], health analytics [60] and autonomous driving [61, 62]. The term cloud is generically used to describe a

remotely located on-demand computing and networking system along with its typical storage functionality. Architectures such as Mobile Edge Cloud (MEC) [17, 46], fog [63] and edge computing bring these resources close to the user to support faster networking and ultra-low latency applications.

Serving IoT devices using edge clouds is proposed in [64, 65, 66] with or without virtualization techniques to provide local compute offload, nearby storage, and networking. Real-time applications such as autonomous driving, traffic monitoring/reporting, and online multi-player 3D gaming have also been considered, [38, 67, 68, 69]. Applications of ICN (Information Centric Networking) have been proposed in [70] as a means to reduce network complexity through named services and content. A three-tier cloud of things (CoT) system is modeled in [71] which identifies edge cloud is a key design element for time-constraint applications. Attempts are also made to provide hierarchical models of edge clouds thereby enabling aggregation capabilities similar to data center networks [72]. Understanding network topology is a critical step in analyzing a cloud or edge network mainly due to effect of routing on latency and throughput. Attempts have been made to characterize the network using geographical properties in [73] using data of autonomous system (ASes) and their relationships, to create a network topology for realistic analysis.

Motivated by faster compute and connectivity needs of newer AR/VR applications, an edge-centric computing is described in [74]. A QoS-aware global optimal edge placement approach is described in [75]. An energy efficient resource allocation strategy is proposed in [76] considering link layer parameters. A small cell based multi-level cloud system is simulated in [77]. Existing literature either relies on a central controller for an optimal edge placement or the use of new network hierarchy to realize improvements in system performance[78, 79]. Studies aimed at determining the overall capacity of a edge cloud system to support multiple applications using a city-scale network are lacking in the existing literature. To the best of our knowledge, this is one of the early attempts to characterize such a hybrid system with respect to edge-cloud resource distribution, inter-edge bandwidth, AP-cloud bandwidth and system load.

2.7 Summary

This chapter provides a framework for modeling and analyzing capacity of a city-scale hybrid edge cloud system intended to serve augmented reality application with service time constraints. A baseline distributed decision scheme is compared with a centralized decision (ECON) approach for various system load, edge-cloud resource distribution, inter-edge bandwidth and edge-core bandwidth parameters. The results show that a core cloud only system outperforms the edge-only system when inter-edge fronthaul bandwidth is low. The system analysis results provide guidance for selecting right balance between edge and core cloud resources given a specified application delay constraint. We have shown that for the case with higher inter-edge bandwidth and edge computing resources, a distributed edge selection achieves performance close to centralized optimization, whereas with ample core cloud resources and no bandwidth constraints, ECON provides a lower average response time. Our study shows that adding capacity to an existing edge resource without increasing internetwork bandwidth may actually increase network-wide congestion and can result in reduced system capacity.

Chapter 3

Techniques to Enable QoS Support in MEC

Achieving advanced MEC services such as dynamic resource assignment and slicing, maintaining Quality of Service (QoS), and enabling heterogeneous virtual functions are some of the technical challenges associated with edge-cloud enhanced 5G architectures now under consideration. This chapter proposes a named-object based virtual network architecture to support low-latency applications in the MEC.

3.1 Introduction

Mobile Edge Computing (MEC) is envisioned to be a core component in future cellular architectures, expected to grow rapidly in the next few years due to continuing large-scale adoption of smartphones as well as emerging technologies such as IoT (Internet-of-Things) and augmented, virtual or mixed reality (AR/VR/MR) [80, 31, 32]. MECs exploit resource locality and have been embraced by the infrastructure and the service providers for their network evolution. In particular, providers are increasingly aiming to distribute their service points of presence (i.e. processing and storage) in order to exploit locality and serve their clients right at the edge of the networks they are connected to. The industry and research communities alike have embraced this approach and are proposing solutions known as edge clouds [81, 82, 33] or fog computing [83] that can better scale and provide low delay services to real-time applications.

Edge clouds distributed at the periphery of the network represent a conceptually simple and scalable solution for delivering computing services to mobile users. Moreover, because of the lower network delay in reaching cloud resources, MEC offers the potential to meet strict service requirements (e.g. low latency). However, this comes at the cost of significant technical challenges associated with moving cloud processing

from a centralized data center to a loosely coupled set of servers located at the edge of the network. One central challenge is that of distributed control; by their very nature, edge clouds are placed in multiple network domains with heterogeneous bandwidth and latency properties without a single point of control. While existing solutions such as network slicing and service chaining provide a means to use distributed, heterogeneous resources, the end-to-end quality remains a concern due to insufficient large-scale telemetry techniques in the currently deployed networks. A second key challenge arises from heterogeneity of computing resources and the limited amount of computational power edge systems can be equipped with. In contrast to the previous data center driven cloud model, edge clouds are often co-located with the existing network equipment and deploy limited computational resources. This implies the need for distributed resource management (i.e. task assignment, load balancing and application-level quality-of-service management) across heterogeneous edge computing resources. Furthermore, network wide changes arising due to the user mobility, link or node failure and network congestion induce additional challenges in maintaining end-to-end service quality.

A general technology and architectural solution for edge clouds will thus require: (a) control as well management plane protocols to provision these heterogeneous resources in real-time, (b) distributed or centralized resource assignment strategies, for traffic load balancing, orchestration of computing functions and related network routing of data, (c) mobility management techniques such as dynamic network slicing, and (d) low-overhead mechanisms to reach to these heterogeneous set of distributed resources in real-time. This work attempts to design, integrate and evaluate these components aiming at fulfilling requirements of advanced services using MEC architecture. In summary, the main contributions of this work are:

- Design of *NOVN**, a Layer 3 virtual network framework that can provide the control mechanisms to connect distributed resources across network domains.
- Starting from the *NOVN* framework, we develop routing mechanisms that exploit the abstractions of the architecture to support distributed edge-cloud services.

*NOVN was jointly developed in collaboration with Dr. F. Bronzino.

This technique, called application specific routing (ASR), supports routing service requests based on cross-layer information extracted from network and application.

- Building techniques to achieve dynamic resource management using in-network resource slicing and devising mechanisms for the quality of service (QoS) control.
- Finally, develop a working implementation of NOVN, ASR and advanced service scenarios using the Click [84] modular router, implemented on the MobilityFirst network architecture software prototype [85]. As part of this effort, we present experimental results obtained to validate the feasibility of this architecture and demonstrate significant latency improvements for real-time applications.

The rest of this chapter is structured as follows. Section 3.2 presents edge cloud requirements to support the advanced services such as cross domain connectivity, dynamic re-routing and cross-layer network support, and align them with the key MEC architectural components. Section 3.3 introduces the core design of *NOVN*; starting from the definition of named-objects, the high level design choices taken are described. The impact of name resolution server (NRS) on the scalability and consistency of NOVN is detailed in Section 3.4. Section 3.5 introduces how *NOVN*, integrated with techniques such as application specific routing and network slicing could be exploited to support advanced network services. To support the proposed design, in Section 3.6 a comprehensive set of experiments based on a working prototype deployed on the ORBIT testbed [86] is presented. Performance evaluation of the proposed NOVN architecture and the related scenarios including comparison with the overlay based solutions are described in the Section 3.7. Finally, in Section 3.8 a further discussion on the design choices and a comparison to related work is provided. Section 3.9 concludes the chapter.

3.2 Edge Cloud Requirements

In this section we discuss the requirements imposed by edge clouds on developing an architecture involving virtual network and advanced MEC services to interconnect and manage distributed resources. Starting from a review of existing virtualization techniques, we discuss the need for the introduction of in-network Layer 3 and service

virtualization.

Edge clouds are highly distributed architectures that require loosely coupled coordination mechanisms to operate. Resource allocation in edge clouds is more difficult than in a data center. This is due to the fact that edge clouds do not have the law-of-large-numbers advantage of a data center which aggregates requests from tens of thousands of users. Instead, they must deal with requests from smaller numbers of users characterized by significant randomness in both the spatial and temporal dimensions. Due to their physical presence in multiple network domains and the type of resources they deploy, the following requirements are identified in contrast to the ones usually presented by datacenter based clouds.

Cross Domain Connectivity. Management of distributed cloud resources becomes more complex when the edges extend across multiple domains. A key requirement for this scenario is to be able to synchronize resources to coordinate and communicate state potentially across multiple domains managed by different commercial entities such as network or service providers.

Dynamic Re-Routing. Due to the nature of IP addresses, any configuration change cased by failure or resource migration requires to reconfigure connectivity between edge computing resources. The new information has to be propagated across all the participating entities. This can – and often does – cause all ongoing traffic to be lost. This is due to packets not being able to carry the necessary information to self-correct temporary errors. Approaches to reduce this impact have been explored [87, 88], but require the creation of dedicated control channels to maintain persistent traffic flow.

Support Cross-Layer Interactions. Edge clouds require dealing with a mix of computing and networking resources with complex cross-layer interactions and considerable heterogeneity in both networking and computing metrics across the region of deployment. Conventional large datacenters have addressed this problem by requiring uniformity in the network fabric and using software-defined network (SDN) technologies to assign resources in a logically centralized manner. On the contrary, for a distributed architecture, a key requirement arises due to the need of dynamic allocation of cloud

processing requests across available edge computing and networking resources [89].

Seamless Service Integration. Edge clouds promise to support tighter close loop low latency applications which require a seamless integration of services with the network entities whose performance can be monitored, reported and enhanced. The key requirements therefore is to design mechanisms which can blend service state parameters into the network to create a fully virtualized end-to-end QoS-enabled system.

The mapping of these requirements to the corresponding architectural component of MEC is further discussed as follows.

3.2.1 Enabling QoS using Virtual Networks

Virtual Networks (VNs) have been proposed as a means of connecting resources across the Internet, supporting the illusion of a customized network with user-specified topology, security and performance characteristics matched to application requirements [90]. Depending on the purpose, different techniques have been applied at different layers of the networking stack in order to realize virtual networks. Cloud networks have been one of the main adopters of virtual networks, with VN techniques being used to abstract the distribution of physical and logical resources - e.g. applications, databases and more - within data centers, allowing for flexible management techniques [91, 92]. Thanks to the simplicity of the solution, together with new technologies like SDN, LAN based VNs allow for a powerful and efficient framework for coordinating resources within a data center.

While VLAN based solutions employed within data centers could be considered to solve this challenge, they cannot scale outside of a single domain. Existing solutions that can work across domains either only support point to point connectivity between remote cloud locations [93, 94] or are based on overlay solutions (e.g VINI [14]). Overlay VN solutions, while flexible, may incur high overhead and lack the visibility of underlying network layer performance parameters, limiting their utility in scenarios that might benefit from custom metrics and deeper cross layer optimization [95, 96].

Existing VN solutions can be roughly grouped into two categories: tag based virtualization at Layer 2 and overlay based Layer 7 solutions.

Tag Based Virtualization. Tag based approaches exploit flat unique identifiers placed at different layers of the network stack to uniquely identify packet flows. Example of this are VLANs [12] and MPLS [13]. Cloud networks have been one of the main adopters of Layer 2 virtual networks, with VN techniques being used to abstract the distribution of physical and logical resources - e.g. applications, databases and more - within data centers, allowing for flexible management techniques. This approach is exemplified by NVP [91] (and similarly by FlowN [92]) that exploits it to implement a network management system, within an enterprise data center. The core issue with these solutions is the limited scope in which they can be applied, as the employed tags are limited in size and have validity only within a single network. For this reason they can solely be used to support single domain solutions.

Overlay Networks. Overlay networking approaches, e.g. VINI [14, 26], represent a flexible way for deploying experimental networks and protocols on top of the existing infrastructure. Through encapsulation of network packets on top of UDP packets and tunneling across participating nodes, they allow for the quickest solution to implement experimental protocols on top of the existing infrastructure. With this solution, flexibility and simplicity come at the cost of additional overhead. Moreover, residing at the application layer they lack visibility of the underlying network environment, not providing support for the aforementioned cross-layer interactions.

The need for Layer 3 network virtualization

Looking at the two available solutions, we identify three limitations: 1) Most virtualization techniques are limited to single domain scopes, e.g. a data center or an access network, 2) When extended to support larger networks, they either need full control of the network environment, or 3) they rely on overlay solutions that are expensive due to the generated overhead and lack any access to the underlying network environment.

The overall goal is to provide a solution that enables the exchange of information between the virtualized environment, the applications that run on top and the underlying network. This solution should offer service providers the ability to exploit network virtualization to enhance deployed solutions like edge clouds, where applications might benefit from affecting routing decisions based on custom metrics and cross layer optimization. From this analysis, we identify the network layer as the right level to host a Virtual Network design. Layer 3 is by definition where protocols are used to interconnect networks resources. Extending it to support virtualization provides the most natural solution to conveniently support interconnecting resources that span multiple networks. The next section defines how a VN can be integrated into the network layer.

3.2.2 Application Specific Routing

Building on top of the L3 virtual networking concept, attempts are made to enhance applications by allowing them to provide hints to the network to optimize routing decisions [95, 96, 97] by implementing a novel technique called Application Specific Routing (ASR). ASR offers applications a solution for pushing small snapshots of compute status data into the virtual routing fabric providing a control environment for distributed services on top of limited edge resources. For example, consider a mobile edge cloud scenario where the application goal is to connect mobile devices to the "best" edge cloud server: while in a normal networking environment "best" might correspond to the "nearest", in heterogeneous environments, varying computing loads might require delivery to a lightly loaded cluster which is not necessarily the closest one in terms of network distance. Through ASR, the architecture can support advanced *anycast* delivery service allowing virtualized routers to consider application status and perform custom routing decisions.

3.2.3 QoS Control

Traditional QoS control mechanisms require complex packet sniffing and processing to manage network resources for achieving traffic prioritization and resource reservations.

The name-based network virtualization technique described in the next section simplifies the QoS control by mapping each virtual network to a unique identifier (Virtual Network Identifier – VNID) encapsulated in the packet header. Striping VNID from the packet header and querying a Name Resolution Server (NRS) for the VNID's allowed traffic capacity provides a L3 in-network support for the scanarios such as shaping network traffic, limiting bandwidth for a VN, and ensuring QoS using the ASR metrics.

3.2.4 Network Slicing

With a network entity (NE) such as router allowed to be a part of multiple name-based virtual networks, network slicing is achieved by the statistical multiplexing of the resources while pushing the VNID to the resource mapping to the NRS during VN instantiation or dynamically, and run-time retrieval of the same by the NE involved in the VN. The prime advantage of this approach is that the resource provisioning and their chaining need not be done in advance. At each NE hop, both the resource metric and the next hop information is obtained from the NRS which thereby inherently handles scenarios such as node failure, link failure and traffic congestion without affecting the ongoing virtual network connection.

3.3 Named-Object Based Virtualization

Recognizing the need to provide a solution that offers the logical simplicity of L2 network virtualization while offering the flexibility to control traffic across network domains, this work presents *NOVN* [98], a virtual network solution that exploits the concept of named-objects [99] introduced in the MobilityFirst future Internet architecture [16] to realize a logically clean, easily deployable, virtual networking framework at Layer 3. *NOVN* tackles the control mechanism challenge by applying name indirection to create clean partitions across logical layers (Figure 3.1). First, physical network resources are mapped to globally unique names, eliminating the need of continually tracking routers addresses and possible configuration changes. A second layer of abstraction then maps network elements to the participants of the virtual network, creating a logical network

on top of the infrastructure.

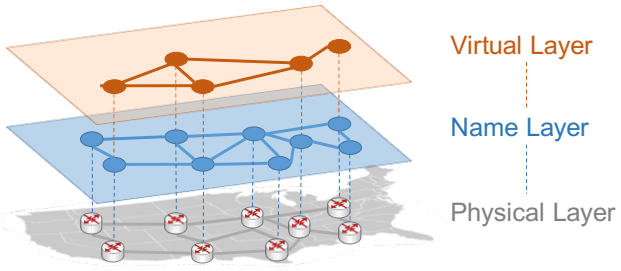

Figure 3.1: *NOVN* layers of abstraction.

For more than a decade, the research community has advocated the separation of names (identities) from network addresses [100, 101, 102, 103, 16]. Named-objects [99] are a powerful abstraction achieved through the use of a dynamic globally available Name Resolution Service (NRS) for mapping names to routable network entities. This separation has inherent benefits in handling mobility and dynamism for one-to-one communications. The general concept of named-objects can be extended to achieve considerable flexibility in creating a variety of new service abstractions [104] as shown in Figure 3.2. First, names can be used to represent many different Internet objects; for example, a cell-phone, a person, or a group of devices; the latter concept also applies in the context of network virtualization, as it provides the basis for *NOVN*'s solution of defining participation of network elements to the logical network. In this case, the named-object abstraction can be used to define entire VNs and store the corresponding topology directly into the NRS. The routers' job is then simplified as they can support multiple virtual network policies simply by indexing their routing table to the Virtual Network Identifier (VNID) associated with a given network. This makes it possible to operate VNs without the need for any additional overlay protocols, creating the sense of VNs as an integral feature of the network protocol stack. The following sections provide the general concepts of how this process is defined.

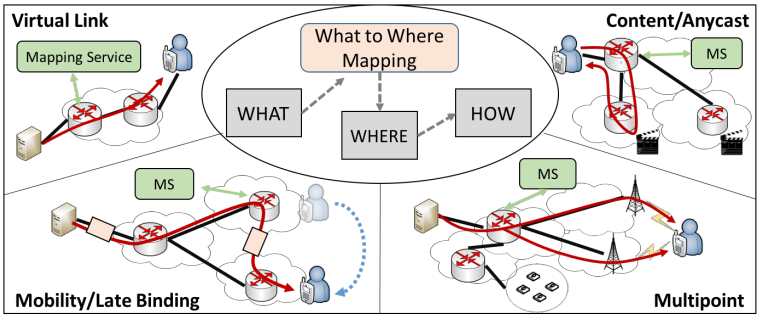

Figure 3.2: The *named-object* abstraction applied to core use cases.

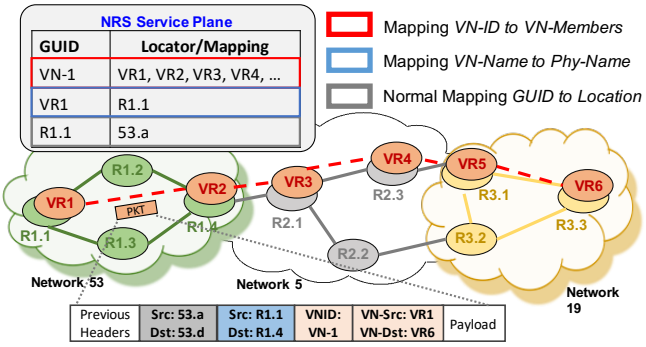

Figure 3.3: NOVN design

3.3.1 NOVN General Design

NOVN addresses the fundamental issues of virtual network management and deployment support through the use of named-objects. Figure 3.3 lists for clarity the set of core design operations are at the base of the framework. To simplify the discussion, three basic assumptions are considered throughout this section: (1) the availability of a globally accessible NRS capable of storing mappings from *names* to list of *values*; (2) the ability to identify network classes based on a unique identifier (SID); and (3) the flexibility of accessing names and addresses as part of a network header to enable hybrid routing, similar in spirit to the one employed in the MobilityFirst architecture [16].

Logical Definition of a VN through Naming. *NOVN* simplifies the definition of the virtualized logical layer through information offloading to the NRS. This is done as a three step process: 1) first, a unique identifier is assigned to the VN and a mapping from

such name (*VNID*) to all participating resources is stored in the naming service (red box in the Figure); referenced resources are identified with a name that has meaning only within the limits of the VN logic - i.e. they are unique and not shared across different VN instances; this provides the dual function of simple access and distributed information recovery. 2) Each VN resource name, is then mapped into two values: a) the name identifying the resource the virtualized element is running on top and b) the list of its neighbors. 3) Finally, these identifiers are mapped into physical Network Addresses allowing for normal forwarding operations. Items 1 and 2 above define the higher abstraction level shown in Figure 3.1 and their mapping into the mid-layer, while item 3 provides the last translation to the bottom layer, that is, the physical infrastructure.

Bootstrap Process & Management. As the topology information is made available at a global scale through the NRS and can be dynamically retrieved from participating resources, the scope of what information is required to share at each layer of the network infrastructure is limited in comparison to other solutions, e.g. [14]. This allows two core issues to be handled separately: the *local* problem of mapping virtual to physical resources and the *global* problem of coordinating the virtualized logic across domains. The first one can be handled either in a network-by-network basis or by a centralized authority while the second one is offloaded to the NRS. To this end, the bootstrap process in *NOVN* is then limited to allocating on participating nodes instructions on how to retrieve the VN topology, i.e. the VN unique identifier used to query the NRS, and the information about the physical resources that are required. Similarly, management operations, e.g. migration, of resources can be handled through NRS offloading too, whereas local changes are reflected into the globally accessible service and dynamically resolved at forward time.

Routing & Forwarding. Providing full flexibility for different routing configurations, *NOVN* does not constrain VN users to employ specific routing protocols. Routing information is exchanged across nodes through control packets encapsulated accordingly

in order to reach participating nodes. Similarly, data forwarding happens on a hop-by-hop manner across routers of the virtual network. When a data chunk reaches one of these routers and a routing decision is taken, the chunk is encapsulated within an external network header that contains information to reach the next VN router (shown in Figure 3.3). At nodes not participating in the protocol, normal routing decisions are taken using the external network header. As names identify each hop, forwarding can happen independently from the physical network configuration.

3.3.2 An Embedded Virtualization Abstraction

Conventional network virtualization techniques suffer from the fundamental shortcomings of the underlying IP architecture and address structure, limiting their flexibility and increasing deployment complexity. Consider the case of overlay based solutions (e.g. VINI [14]) where virtual router interfaces are assigned private IP addresses and then mapped to public ones that can be used to tunnel packets across participating resources (Figure 3.4). Due to the nature of IP addresses, any configuration change due to failure or resource migration requires the tunnel to be reconfigured, the new information to be propagated across all the participating resources, causing the loss of all ongoing traffic. This is due to packets not being able to carry the necessary information to self-correct temporary errors. Approaches to reduce this impact have been explored [87], but require the creation of dedicated control channels to maintain persistent traffic flow.

NOVN solves this issues by creating clean partitions across logical layers, as previously shown in Figure 3.1. This is obtained by recursively mapping from VN dedicated names, to network elements names and finally to the physical addresses. These layers of abstraction are critical in allowing a separation of management issues. Consider, for example, the case of virtual router migration. In *NOVN*, the process is simplified by limiting the impact of the migration to remapping identifiers between the top two layers. Once the required migration process is defined, the entry mapping the VN element to the network element is re-written to the new location. If in-flight packets are forwarded during the transfer process, name indirection allows for fast recovery without

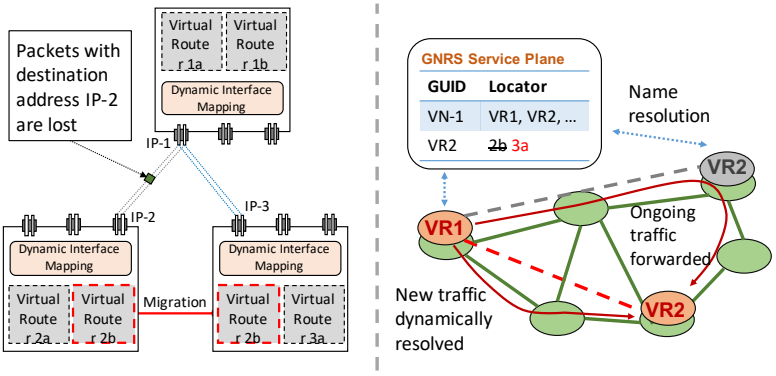

Figure 3.4: The effect of router migration on overlay deployments (left) and *NOVN* (right).

need of end-to-end retransmission, by resolving the delivery location through the NRS. Similarly, if a physical machine needs to be replaced due to failure or an address change is required, a new one can be instantiated and the state transferred.

One could argue that the employment of multiple layers of abstraction can introduce additional overhead due to the resolution costs of crossing the different logical layers through name resolution and due to the additional headers employed. The impact of these is alleviated though by the employment of two separate techniques: 1) While name resolution can become costly if performed for each forwarding decision, the action is not required as for the majority of the time the resources do not change; hence, information can be pre-cached on the participating routers and only once resources are notified of occurring changes they have to update their mappings by querying the NRS. 2) As tag switching and SDN techniques [105] have demonstrated, matching multiple fields in hardware is a feasible task and as software components take over, this becomes an even easier task. An empirical demonstration of the feasibility of the approach will be given as part of the prototype deployment presented in later sections.

3.3.3 Separating Local and Global Tasks

Managing resources in virtualized environments increases in complexity when extended to multiple domains. This is true for overlay approaches, where resources need to be

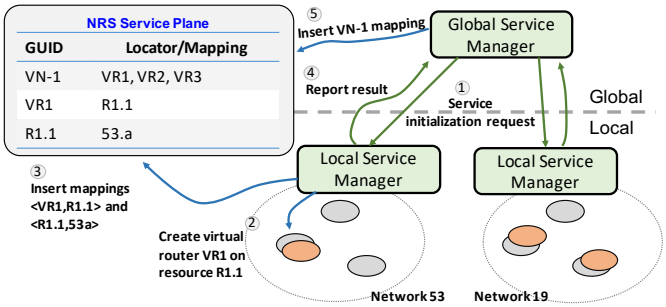

Figure 3.5: Separation of local and global scale problems through a distributed coordination plane.

coordinated and communicated potentially across multiple networks in order to synchronize, and it is mostly untreatable for tag based solutions that are usually optimized for small domains, e.g. a data center or an access network [91]. This is a consequence of the complexity of assigning coherent resources across multiple domains that can be managed by different commercial entities.

NOVN approaches the problem by creating a distinction between the *local* problem of assigning network and computing resources and the *global* problem of providing coordination mechanisms across domains. The NRS and the named-object abstraction are the key elements employed to offer ways for eliminating the complexity as they provide the infrastructure a way to offload the sharing of the virtualized topology and the mapping of the underlying elements. With this, network administrators can then separately focus on deploying techniques for optimizing the management of their infrastructure and the placement of the resources while relying on globally available mappings for coordinating with partnering networks.

Figure 3.5 outlines the resource allocation process when a hierarchical set of service coordinators is employed. In this example, each network domain exposes an interface that services deploying a multi-network VN can invoke to allocate resources that span across the participating networks. While this example employs the concept of a single service interface per network with a centralized controller for requesting and coordinate resources across networks, the same tools can enable more distributed mechanisms for allocating and deploying virtual networks.

3.3.4 Network State Exchange

Similar in spirit to previous attempts of providing full control of the deployed routing protocols on top of virtualized networks [14], *NOVN* has been designed to offer routing independent network abstractions. In other words, administrators of virtual networks can independently choose which routing protocols better suit their needs as long as they have ways of learning the underlying network conditions, e.g. virtual links costs. This latter problem could be approached in multiple ways: a) resorting to over the tops approaches where measurement tools are used to extract the information, as done in VINI [14]; b) by allowing routing information sharing across layers, through the use of APIs exposed by the underlying networking logic. The current *NOVN* design favors the second approach, acknowledging the increasing reliance of software based routing tools that can support APIs used by the virtual layers on top to extract link state information.

3.4 Name Resolution Service Impact on the Architecture Scalability

The named-object abstraction of *NOVN* relies upon the use of a Name Resolution Service (NRS). Therefore, the performance of the NRS becomes critical to achieve consistent performance. Multiple previous projects have demonstrated how different NRS designs [106, 107] achieve low resolution latency goals of less than 100ms on average for lookup operations. Moreover, additional studies demonstrate how to further reduce response time exploiting concepts such as caching and locality [108]. Commercial [101] and experimental [107] versions of such services are currently running and are available for use. This section provides an overview of the different implementation approaches and details how to handle consistency and scalability issues while relying on the NRS to deploy the NOVN architecture.

3.4.1 NRS Implementations

NRS designs can be classified as either hierarchical or flat based on their naming structure. For hierarchical namespaces, commercially proven implementations such as DNS

are available. Unfortunately, it has been demonstrated that DNS is not suitable as an NRS implementation in a highly mobile environment due to its static placement strategies inherently, e.g. time-to-live (TTL) based caching, limiting its effectiveness upon end-host mobility. This work relies on the use of a flat namespace for which there are several NRS implementations available in the literature. We select two designs as candidate implementations for NOVN.

Auspice [107]. Auspice's logic uses a demand-aware replica placement engine to distribute GUID to NA records across available caches to provide low lookup latency, low update cost, and high availability by carefully choosing the number and locations of required replicas for each GUID as per the lookup and update request rates, the existing replicas for a GUID, and aggregate load at a replica. This geo-distributed engine is implemented as a logically centralized authority which tracks query demands using a recursively mapped key-value store. In Auspice, a GUID belongs to a number of replica-controllers (fixed) and active replicas (variable). The replica-controllers maintain information about active replicas such as their number and locations whereas the actives replicas maintain a GUID record and process a request. The placement algorithm is computed locally at each replica using lookup to update ratio of a GUID thus limiting the update cost. The replica location is chosen such that the lookup latency is minimal by selecting some replicas closer to the higher demand zones while others placed randomly for load balancing.

DMAP [106] / GMAP [109]. In DMAP, name mappings to network addresses are distributed among participating Autonomous Systems (ASes) while also choosing a deputy As which has minimum IP distance to the current hash value of an IP address. DMAP routers apply K consistent hash functions (where K is the number of replicas desired) to map names to the gateway routers wherein they are are stored. GMAP builds over DMAP by organizing the name to NA mappings hierarchically in three levels – local, regional, and global – to exploit spatial locality. Furthermore, the server lookups are load balanced using a concept of probabilistic caching, thus improving scalability over the baseline solution.

3.4.2 NRS Challenges

The implementation of a large scale database such as a Name Resolution Service creates challenges of information freshness as well as lookup delay that might compromise the requirements of the NOVN architecture. In the following section we describe how both architectures could be safely employed to deploy NOVN.

Consistency. Inconsistency may arise when a query reaches a cache that does not hold an up to date name/address mapping due to host mobility and late update, or incorrect prefix cache in a BGP table, thus incurring additional query response delay.

In Auspice [107], the consistency issue is handled using an explicit coordination mechanism between the consensus engines of the replica-controllers and active replicas, where each NRS node propagates information to a set of replica servers. In DMAP [106], consistency is quantified as the probability of BGP churn by varying the percentage of prefixes that are newly announced or withdrawn from 0 to 10%. If a GUID is not found at an AS, it replies with a "GUID missing" message and the querying node contacts another replica. The fragmentation of IP address space may lead to an unannounced IP as a hash causing the IP hole problem. To maintain consistency, the withdrawing AS sends a GUID insert message to the deputy AS while deleting its own entry. Later, the subsequent queries hit a IP hole and thus the deputy AS is queried to obtain the latest mapping. In case a query is not found at an AS, the querying AS sends a one-time migration message to the deputy to self-assign a mapping thus removing an inconsistency.

GMAP [109] further enhances DMAP's consistency mechanisms by using a sequential scheme which piggybacks the server availability updates in the query replies along the request path. It is shown that for up to 5 replicas, there is only a 5% failure rate which shifts the median from 40.5ms to 41.3ms which is acceptable for consistency in NOVN as there are practically a very few origin changes for an AS prefix according to [110, 111].

Scalability. An increase in number of replicas improves NRS scalability but creates a consistency problem. In Auspice, the dynamic nature of its placement algorithm

maintains a balance between the cost and the performance. At lower loads, the lookup latency is minimized by selecting maximum number of replicas whereas at higher loads, only the popular GUIDs are replicated at multiple locations thus keeping cost under control without sacrificing performance a lot, making the solution scalable.

In DMAP the balance between consistency and scalability is provided by (a) using a single overlay-hop path to a storage location and (b) not adding a table maintaining traffic unlike other DHT implementations. The query response time evaluation of 10^5 name insertions and 10^6 queries shows that with K=5, 95% of the queries complete within 86ms which is reasonable for a large scale system. The query response delay in DMAP is low because the updates do not introduce additional delays, and a name/address mapping is stored at multiple locations which can be queried by a node from a location closest to the itself. GMap provides scalability at the cost of not maintaining per-GUID state at all the servers, and keeping the cache size low for popular GUIDs. As even commercially available VN techniques [112] introduce a delay in the order of ∼150-200ms when using 3–4 network hops —almost twice as compared to the delay value of less than 100ms achieved by DMAP— we argue that these implementations are acceptable for NOVN.

3.5 Advanced MEC Techniques

The increasing softwarization of the network infrastructure, enabled by the advancements in computing power and virtualization techniques, has facilitated the support of new applications and services inside the network. Among different opportunities, network vendors and researchers have looked at solutions that explore how to better integrate inputs from the application logic to optimize network functionality [95, 96, 113, 97]. While this is a useful direction, more general solutions capable of extending beyond the limits of single networks are still lacking. Such working solutions would highly benefit distributed service scenarios, where advanced control mechanisms are required. In this section, techniques to efficiently utilize the named-object based virtualization in the MEC architecture are described.

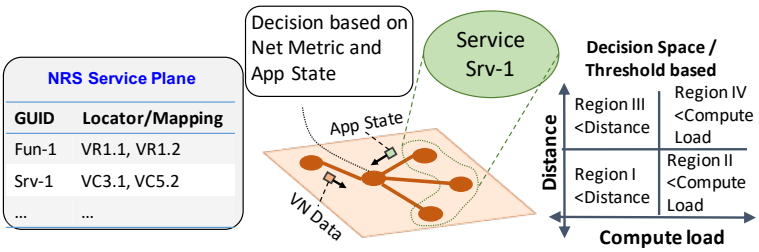

Figure 3.6: Application Specific Routing as an advanced routing service for the edge cloud use cases

3.5.1 Application Specific Routing

MEC architecture using NOVN is extended to support advanced routing through a technique called Application Specific Routing (ASR). ASR defines a mechanism aimed at exploiting a comprehensive set of information from both network and application layers to enable custom delivery mechanisms, giving service providers the flexibility to incorporate parameters which allow for utilizing information above the network layer for routing decisions. Consider, for example, the case of a service deployed at multiple locations across different domains: application state could be exploited to implement advanced *anycast* delivery based on network metrics and service load at the end points.

Two key technology components are required and introduced into the *NOVN* framework to support ASR: (1) the ability to aggregate multiple service instances under a single name, a natural extension of the named-object abstraction. (2) the ability to make application nodes participate in the routing protocol by sharing their application state. *NOVN* supports the first one by offloading the list of participant locations under a single name into the name resolution service and the second one by allowing custom routing protocols to be deployed on top of any underlying infrastructure and integrating end point APIs to push application state into the VN.

For edge clouds to scale well and deploy easily, it is necessary to develop a robust and self-organizing distributed architecture analogous to the way in which inter-domain protocols in the Internet enable networks to cooperate on routing while retaining some measure of local policy control. Of course, the distributed algorithm design problem

for edge clouds is a more difficult one because we are dealing with a mix of computing and networking resources with complex cross-layer interactions and considerable heterogeneity in both networking and computing metrics across the region of deployment.

ASR supports edge cloud solutions through the support of advanced cross-layer routing mechanisms. Consider for example the scenario depicted in Figure 3.6, where a collection of servers offer a service to its clients. *NOVN* and ASR provide the base to deploy such distributed tools by: a) allowing push of state to participating nodes and b) make use of the named-object abstraction to support advanced anycast delivery to service instances based on both network and application metrics (Figure 3.6). At branching locations, routers can then take informed decisions. For example, Figure 3.6 shows a decision space scenario where given thresholds define different states that can influence how routing decision. While the effectiveness of the ASR approach has been proposed in our previous work in the context of cyber physical systems [114], coupling *NOVN* with ASR can support a low latency and scalable solution for any service that would benefit of the locality of edge clouds.

3.5.2 Quality of Service Control

Maintaining QoS is a key requirement in the MEC architecture specifically to support low-latency applications. General QoS control mechanisms favor class-based approach where each traffic flow is assigned a QoS class identifier (QCI) to tackle issues such as admission control, queue management, and limiting bandwidth. The QCI value is pre-configured and cannot be adjusted dynamically during the run-time therefore lack required flexibility in the QoS control. Edge clouds are often co-located with the existing network equipment and often have limited computation and storage. For this reason, they are solely capable of hosting a limited amount of applications at any point in time, requiring service orchestrators to engage in dynamic traffic management. *NOVN* maps the VNID to its control parameter by querying NRS at the run-time, thereby providing a per-flow as well as per chuck based fine-grain QoS control. Figure 3.7 illustrates a traffic shaping example using NOVN. A service ID (SID) can be classified either as a part of virtual network or as a best effort traffic at the router during run-time. The

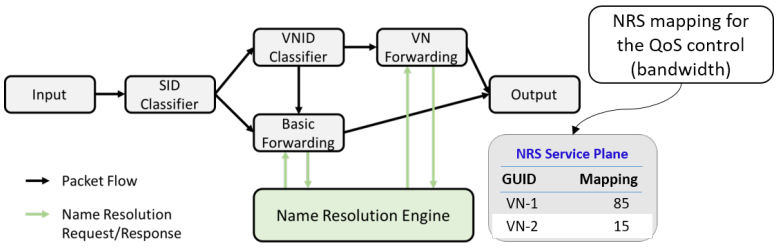

Figure 3.7: QoS control (traffic shaping) example in NOVN

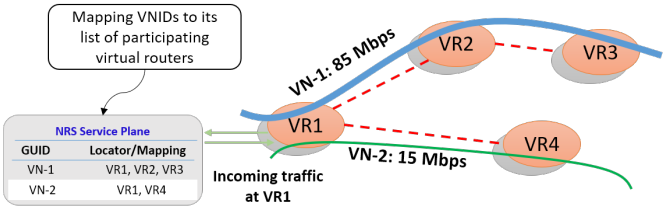

Figure 3.8: Network Slicing in NOVN

VN forwarding engine queries the NRS service plane for the traffic rate information for the VNID and thus achieves the traffic shaping function inherently.

3.5.3 Network Slicing

In a multi-provider network scenario, to support a variety of services, network slicing allows statistical multiplexing of the available resources. NOVN allows a virtual router to be a part of multiple virtual networks thus enabling network slicing implicitly. The resource provisioning to each of the network slice is similar in spirit to the QoS control mechanism described above. Figure 3.8 illustrates as example of two network slices with a common virtual router (VR1). The VNID header lookup at the NRS provides the information about the participating virtual routers. Finally, each of the VN traffic is handled according to its own QoS policy thereby ensuring a cleaner approach to a sliced network.

3.6 Prototype and Experiment Set-up

In order to understand the achievable performance and feasibility of the proposed *NOVN* architecture and its associated advanced techniques, a fully working prototype of the framework has been implemented. The *NOVN* prototype uses as its foundation the MobilityFirst (MF) future Internet architecture [16] prototype [85]. The MF architecture is an example of how the named-object abstraction could be integrated into an Internet network design and for this reason provides the perfect environment to natively deploy the features at the base of *NOVN*. At the core of the architecture is a new name-based service layer which serves as the "narrow waist" of the protocol stack. The name-based service layer uses flat Globally Unique Identifiers (GUIDs) of 160 bits to identify all principals or network-attached objects. Names are resolved through a Global Name Resolution Service (GNRS) that provides APIs to insert and query for <*key,value*> mappings and support hybrid routing schemes [115] that exploit availability of both names and addresses in the network header for dynamic resolution of destination locations. A Service Identifier (SID) flag placed in network header allows network components to be aware of different service types in order to apply different forwarding modes.

The main components of the architecture prototype are three: a Java based GNRS that uses DMap's [106] DHT based implementation to distribute mapping entries, a software router implementing MF's hybrid name/address routing logic and a host guid based API and network stack [104] to run applications on the architecture; while the complexity of the individual components of the prototype is not irrelevant, due to space constraints this work solely focus on the components that have been extended to support *NOVN*'s design, referring to previous work [85]. The open access code repository and code wiki are also fully available for more details [116].

3.6.1 Core Prototype Components

Routers. The software router is implemented as a set of forwarding elements and table objects within the Click modular router [84] run at user-level. As a baseline, the

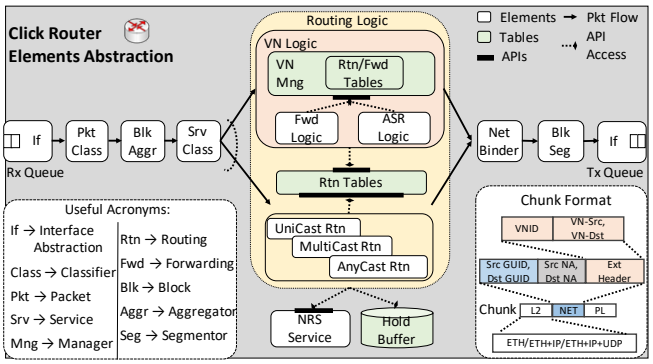

Figure 3.9: Click router elements graph for data plane flow

router implements dynamic-binding using GNRS, hop-by-hop reliable transport using a HOP [117] inspired protocol (by aggregation and segmentation of large chunks of data), and storage-aware routing [115]. It integrates a large storage, via an in-memory *hold buffer*, to temporarily hold data blocks for destination endpoints during short-lived disconnections or poor access connections. A particular instance of this system, implements what we call an MF access router, a router providing access connectivity to clients.

The base router has been extended to introduce the *NOVN* logic (Figure 3.9). Multiplexing across different delivery services is handled via the Service ID (SID) tag available in the MF routing header (*Srv Class*). Encapsulation of the *NOVN* required headers has been implemented exploiting extension fields in the MF network layer.

When traversing a non-VN enabled router, the SID is not recognized and the data is forwarded based on normal unicast rules. Once packets enter the VN logic layer, the router checks whether a) the packet is intended for itself (destination GUID) and b) if the VN belongs to the ones currently active; the simple field base matching exploits VN native concepts as explained in section 3.3.2 allowing for a performant decision logic, as shown in the results of the next Section. VN tables (Routing/Forwarding/ASR) are stored and quickly retrieved via a Hash Map, guaranteeing high performance; when invoked, the routing logic (and if deployed, the ASR one) can access the information and take fast decisions.

The control plane (not shown in the picture) is handled in similar fashion: the current design implements a Link State like Protocol (LSP), to exchange routing information between routing instances; routers periodically generate and distribute the aggregated cost view of each virtualized link to neighbors that, following the logic of the protocol, store and forward the information. Path costs are extracted from the underlying unicast routing tables (*Rtn Tables*) via APIs. Initialization of the logic for a given VN can be done via two different methods: either statically within the click configuration files using as inject point or based on a managing protocol exposed via the Click software control interface.

Finally, the routers have been enabled with interchangeable *Interface* classes that can adapt to different networking environments, supporting different deployment scenarios; these include: a) native support of the MF protocols on top of a L2 network and overlay support both on top of b) barebone IP network or c) a full overlay solution on top of UDP.

Clients. In similar fashion, the baseline client network stack and API [104] have been extended to support *NOVN* operations including: a) exposure of the required API options during socket initialization (i.e. open) to b) instantiate resources in the network stack and c) encapsulation of messages as required by the protocol.

3.6.2 Extended Implementation for the Advanced Services

The core NOVN prototype is further extended to support QoS control and network slicing by introducing a VNID based mapping technique. An MF chuck consists of number of packets. As shown in the Figure 3.10, resource management is achieved by marking the incoming packets in a chunk and then classifying them according to their VNIDs. The classified chucks are stored into a buffer which are pulled by a bandwidth shaper at a specified rate before aggregating them back as a chunk and sending at the output port. This simple VNID based classification and shaping technique enables NOVN with the resource provisioning and traffic shaping, and therefore aids in the network slicing.

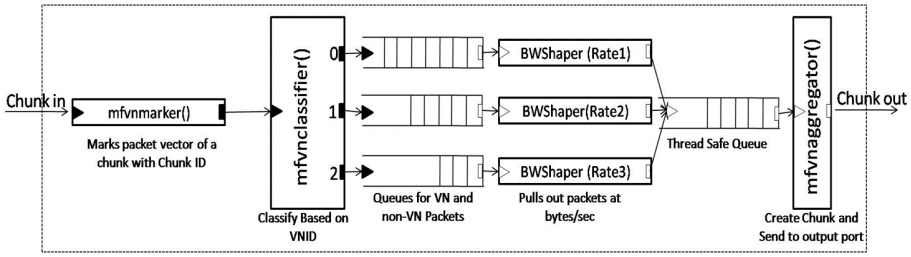

Figure 3.10: A Sample Traffic Shaper Implementation in NOVN

3.6.3 Overlay based VN Implementation

For performance evaluation and comparison purposes, we implement an overlay based virtual network as follows. We integrate OpenVPN [118] based tunnels on top of a barebone IP router implementation using Click [84]. Tunnels connecting nodes are set-up between each pair of virtual routers (VR). Upon transmission, data is encrypted, encapsulated and tunneled to the neighboring virtual router. An encapsulation table maps an UDP tunnel to the public IP of the adjacent router at the overlay virtual router. An OSPF-like (Open Shortest Path First) protocol is used for routing at the IP layer. Predefined virtual paths are set using the aforementioned tunnels between virtual routers. Finally, the VN packet is implemented in click with the following fields: virtual source IP, virtual destination IP, transport identifier (UDP), OpenVPN header, source IP, destination IP and the payload.

The named-object based VN is evaluated by running an additional Name Resolution Service, which for simplicity we deploy using a single server. The network topology information consisting physical router connectivity, physical to virtual router mapping, and participating VN and service identifier, is disseminated at all the routers before the network bootstrap. A named-object VN packet has the following fields: source NA, destination NA, service ID, source GUID, destination GUID, VNID, source VGUID, destination VGUID and the payload, as described in the previous sections. Each virtual router is mapped to its physical router's GUID whose network address is queried from the NRS during run-time.

3.7 Performance Evaluation

A combination of routers and clients have been deployed on the ORBIT testbed [86]. On the testbed, all nodes are interconnected via 1 Gbit Ethernet switches, creating a single L2 network. Selecting 19 nodes, different networks have been deployed for the different use cases analyzed. As the testbed provides a single L2 network, a logical split has been implemented within the click routers to enforce the topology. We present the following evaluation results: (a) a set of micro-benchmark experiments aimed at demonstrating the baseline computation overhead of our VN implementation against the baseline MF prototype, (b) an analysis of how to achieve network slicing in which different VNs can co-exist on the deployed network, (c) results on the traffic shaping to achieve QoS control, (d) an ASR edge cloud use case deployment scenario, and (e) a comparative analysis of NOVN with the traditional VN deployed as an overlay network on top of the current Internet architecture.

3.7.1 NOVN Performance Benchmarks

In order to understand the basic overhead introduced by running the virtual network logic on top of the baseline prototype, two sets of benchmarks are performed: first, a latency evaluation using a *ping*-like application that collects RTTs for a small (64B) and a large (1MB) chunks size; second, using a port of *iperf* that uses the new API and stack to transmit data, achievable bandwidth is estimated. For both scenarios the network shown in Figure 3.11 is used, but traffic generation is limited to VN-2 (blue color).

Latency and Throughput: Total values reported in Table 3.1 account for the sum of three time components: 1) the processing time of the software router (including potentially the VN logic); 2) the queries to the NRS (2ms RTT from the routers to the NRS with query results cached on the routers for 30 seconds); and 3) the HOP like protocol which requires the transmission of initial and final control packets for each chunk to provide a reliable transmission on a hop-by-hop basis. For this experiment, RTTs for the smaller chunk size do suffer some small increase in the *NOVN* case due to the overhead

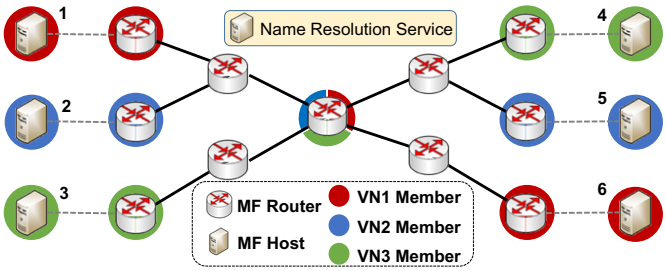

Figure 3.11: Network topology used for benchmarks

Size	**RTT without NOVN**	**RTT with NOVN**
64 B	7.6 ms	8.8 ms
1 MB	128.1 ms	128.1 ms
	Throughput without NOVN	**Throughput with NOVN**
64 B	14 mbps	11 mbps
1 MB	916 mbps	903 mbps

Table 3.1: Latency and throughput NOVN Benchmarks

generated by the processing of the added logic and the additional queries to the NRS (to resolve the higher layer mappings). The effect of the NRS queries is limited though, as they are averaged over the number of total collected samples (1000, one every second), even considering that a 30s cache is quite conservative, especially for VN like scenarios where changes are unlikely to happen in the order of seconds. The bigger size is less impacted by the additional overhead. The performance impact of *NOVN*'s overhead on the achievable throughput is also minimally noticeable, but with increasing chunk size the effect is proportionally minimized. For this metric, the impact of the queries to the NRS is a lesser factor (at 1MB, ∼113 chunks per second are transmitted and only one time every 30s or ∼3400 chunks the NRS is queried). The decrease in throughput has then to be attributed to the additional header and processing overhead caused by the VN logic. Even though these do factor for a decrease in performance, this is small enough that the evaluated scenario does not causes concern for the effectiveness of the design.

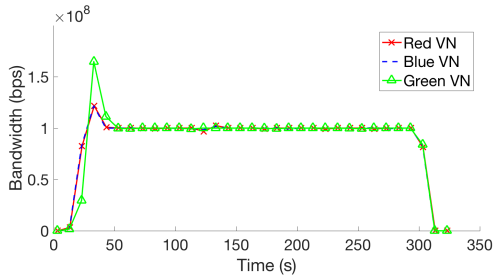

Figure 3.12: Multiplexing NOVN benchmark

3.7.2 QoS Control

A big advantage inherent to the *NOVN* design is the possibility of performing multiplexing across different VNs by natively switching traffic based on a single header field, i.e. the *VNID*. To test the overhead and functionality of the VN switching mechanisms in the prototype, three VNs have been deployed on the network shown in Figure 3.11. Best effort and managed traffic scenarios are evaluated without and with the QoS control mechanisms.

Multi VN Co-existence: Each traffic source (nodes on the left side), generates traffic at 100 Mbps. Figure 3.12 shows the results after running a five minutes experiment without employing traffic shaping. While initial competition on the wire, causes some overshooting of the goal throughput, the traffic stabilizes shortly after and it is maintained until the experiment is completed (at around 300s). The overshooting is introduced by the chunk base nature of the protocols implemented, where a sudden arrival of large chunks (1MB) requires time to adjust.

Managed Traffic Network Slicing: Using the topology described in Figure 3.11, traffic is generated at the rate of 100 Mbps at all three sources and managed in-network using the traffic shaper. VNID to allowed traffic rate mapping information is updated at the start of the experiment and dynamically retrieved during the run-time by querying NRS. As shown in Figure 3.13, each of the red, blue and green VNs pushed traffic up to their allowed limits of 0.5, 10 and 20 Mbps respectively. Similar to our previous observation, while the initial competition on the wire shoots up the traffic, due to the

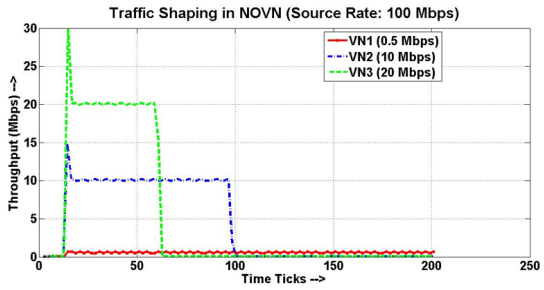

Figure 3.13: QoS control benchmark in NOVN

rate limiting implementation in the traffic shaper, all three VN's traffic stabilizes to reach up to their allowed capacity.

3.7.3 ASR Use Case

To exemplify the implementation of the ASR concept, a closed-loop (round-trip) application has been deployed on the network pictured in Figure 3.14, where clients send requests of 10KB each in size to a set of two servers representing a cloud service. ASR is deployed to consider in its forwarding decisions both network metrics used in the normal routing scheme (latency and delay) and the servers load. Cloud servers loads are emulated by adding emulated delays before sending responses of 10KB back to the client. Server-1 has dynamic load chosen uniformly every 30 seconds from the set of values 0, 0.2, 0.4, 0.6, 0.8, representing linearly increasing delays of 0, 20, 40, 60, 80 ms. Server-2 is statically configured to always select parameter 0.4. A 20 ms extra RTT has been added in the path to the bottom server by using *tc* to emulate different path distance between the servers. Servers announce their load via the ASR protocol every 2 seconds. Figure 3.14 shows the performance obtained, representing the taken decisions by the ASR logic; at the bifurcation, requests are forwarded based on a simple threshold logic, where potential destinations are divided into a decision space in which different regions have higher priority: if there are servers with load lower than 0.5, choose the one with the best path; otherwise simply choose the best path. This guarantees for the experiment setup that all requests are sent to a router with load lower than 0.5 capping response time to $\sim$70ms.

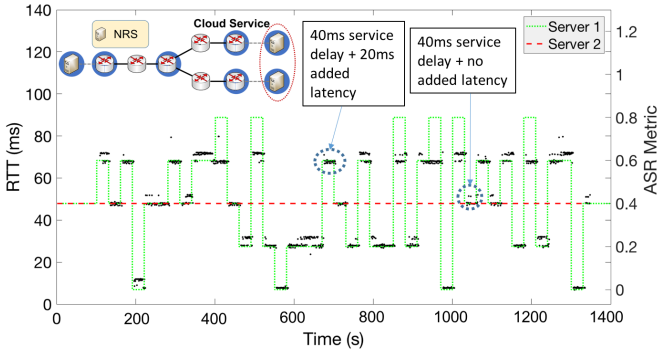

Figure 3.14: ASR edge cloud use case example

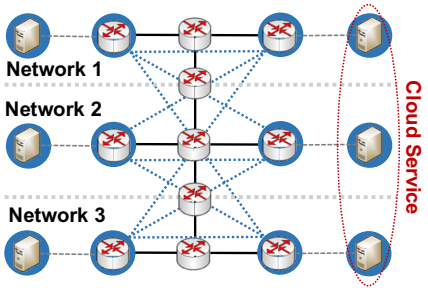

Figure 3.15: Network topology used for edge cloud deployment

This setup has then be extended to represent a more realistic scenario as shown in Figure 3.15. In this case, three clients are deployed, connecting to three networks each equipped with a local service instance. Crossing border routers introduce a 5ms delay each way, replicating the cost of traversing across domains. The server loads are dynamic with the same parameters. Each case has been run for one hour and collected results show how the combination of *NOVN* and ASR impact the service response time. Figure 3.16 shows the obtained results. The following should be observed: 1) up to ∼50ms, the difference between the two lines should be recollected to the local servers' load variations over time (i.e. if the load is below 50%, the local server is chosen) and should converge over a longer time; 2) the ASR impact is very noticeable above such threshold, where 90% of requests are serviced in less than 68ms, a more than 30% improvement from the baseline case (where the local server is always selected).

The *NOVN* framework, as described in Section 3.3, provides a clean way to define

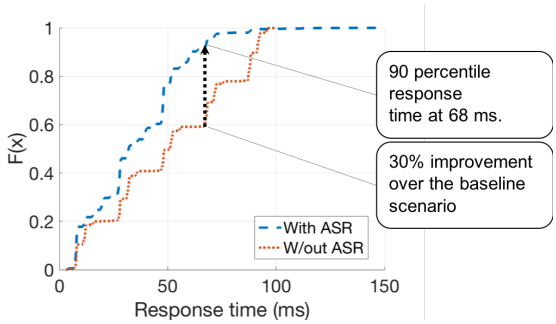

Figure 3.16: Response time for edge cloud deployment

a virtual network topology through the use of the named-object abstraction. While using this technique it is possible to achieve the purpose of providing the high level mechanisms that characterize the system, additional details are required to provide a better sense of how *NOVN* can fully overcome the issues presented and how it could be deployed on top of the current TCP/IP Internet architecture.

3.7.4 Comparing NOVN with Overlay VN Solution

Overlay based virtual networking approaches rely upon complex packet processing at the router and the setting flags to carry extra information such as fragmentation. These approaches increase the round trip time (RTT) of a packet in the network and lowers data throughput, but may also fail the integrity of a tunneled packet for a larger size due to fragmentation flag set. This is generally avoided using a no fragmentation flag which causes loss of packets which are bigger than the MTU. Furthermore, overlay based solutions rely upon tunnels which are set up a priori. In case of a run-time failure, the tunnel needs to be set up again.

During link failures, overlay virtual networks lose packets until the link becomes active again or the route converges, incurring packet loss and lowering system throughput. In case of short duration link failures, the route converges to the same path and therefore the packet loss is directly proportional to the duration of the failed link. For the permanent link failures (equivalently, long duration link failures), the route converges to a different path and therefore the packet loss is proportional to the sum of losses

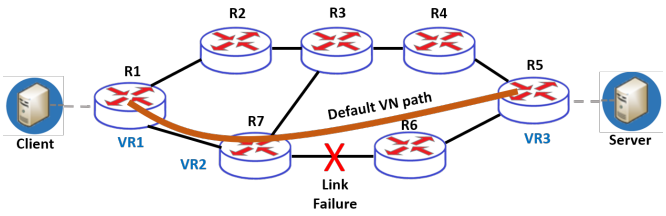

Figure 3.17: Network topology for VN comparison

due to timer expiration and route convergence time. Due to the slow start behavior of TCP, it is time expensive to create new tunnels in case of route change impacting throughput and delay.

In the embedded NOVN approach, network addresses are dynamically retrieved using a logically centralized geographically distributed NRS. The route is therefore resolved at the run-time by querying NRS which strictly decouples network functions from the hardware functions, shifting focus from complex packet processing to a simple packet forwarding. This also alleviates network configuration issues as assigning a GUID to a node is as simple as declaring a variable.

VN comparison experimental set-up We deploy both, the overlay as well as named-object based VN architectures described in the Section 3.6.3 on a small network on ORBIT as shown in the Figure 3.17. Seven routers form the core network and are connected via the Ethernet with 900 Mbps bandwidth. A simple ping application is run from the client to the server with different packet size to compare both the approaches in terms of protocol data plane overhead and recovery time from link failure.

Latency comparison. The round-trip delays associated with the data traversed across the network capture the encryption, tunneling, encapsulation and any other packet processing; therefore, the RTT can be approximated as protocol overhead for the architecture comparison. Table 3.2 shows round trip times (RTTs) obtained for different packets sizes for overlay and *NOVN*. The ping latency is averaged over a large number of pings (>1000). We notice that *NOVN* experiences increased latency compared to

the results obtained by the overlay network. We attribute the added latency to the periodic NRS queries in case of NOVN whereas overlay network is pre-configured and the only overhead it experiences is in replacing headers. Moreover, the MF based solution uses 160 bits long names for objects identification, a large increase in headers overhead compared to the other solution. Even considering these elements, *NOVN* still achieves close performance results compared to the overlay network.

Packet Size	Overlay (RTT in ms)	NOVN (RTT in ms)
64 B	6.1	7.2
1400 B	6.3	7.5
5 KB	7.9	9.6
10 KB	9.4	12.9
50 KB	13.6	17.8
100 KB	20.1	24.4
500 KB	72.4	78.5

Table 3.2: Overhead comparison.

Link Failure. We emulate link failures by introducing packet loss at the link between the routers R6 and R7. We analyse two cases: (i) 100ms (short term failure) and (ii) 100s (long term failure), using the RandomSample element in click router, sampling packets at the loss rate 1 for the specified duration. For the first case, neither of the approaches had enough time to react to the failure and converging to a new path; both cases simply recover once the link is re-established. Packet loss observed in the overlay case is $MTTR{*}rate$ while for *NOVN* there is no loss due to store and forward capability of the router inherited from the MF architecture. In the 100s case, the overlay approach has to wait for the routing protocol to re-converge to a new path and set-up new VPN [119] tunnels before a client can get ping responses back from the destination. In contrast, *NOVN* reacts much faster as the next node's network address is dynamically resolved by querying the NRS. Figure 3.18 compares the effect of link failure for both the cases. The server transmission rate is a ping response to the ten 64 B packet ping requests sent by the client shown the Figure 3.17. The failure is introduced at time t=35 seconds. NOVN recovers in about 1 second without losing any packets due to its in-network store and forward scheme. Overlay VN loses the packets equivalent to the

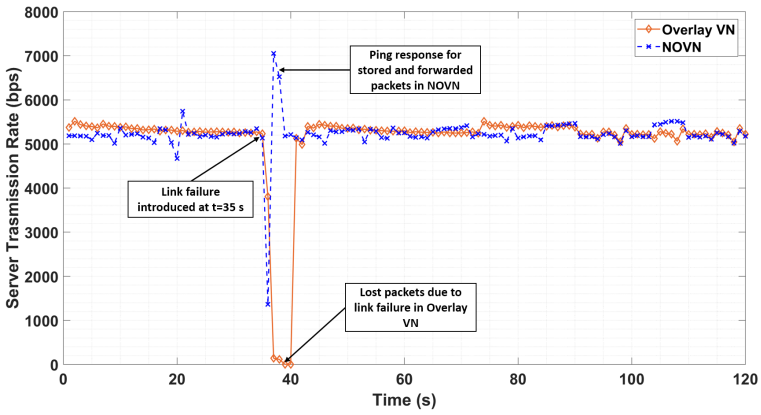

Figure 3.18: Comparing effect of link failure for overlay VN and NOVN

mean time to recover (MTTR) which is more than 5 seconds in this case.

3.8 Discussion and Related Work

Inter-Domain Peering Agreements. Inter-domain connections might require additional coordination across parties involved if no overlay solution is implemented. For this, it is arguable that the increasing reliance of ISPs on point to point agreements via Remote Peering [120] and private interconnections over IXP locations via VLANs [12] would well serve this type of architecture. Both techniques rely on the use of tag based forwarding, e.g. long distance MPLS for the first, to interconnect networks, providing a suitable environment to map higher level VNs defined in *NOVN* to these channels.

Related work. *NOVN* takes inspiration from within two broad categories of works: 1) virtual network designs and management techniques and 2) software based solutions to enhance services on networks. Most recent VN designs in general span from overlay solutions [14, 121] to lower layer integrations using tag switching [91, 92]. *NOVN* differs from all these works by offering a native network-layer solution based on separating names identifying VN resources from the underlying infrastructure. No other work has looked at this type of generalization, providing capabilities that can extend across multiple domains.

ASR takes inspiration from the broad variety of software enhanced solutions aimed at allowing greater control and interaction to application and services populating networks. SDN [105] and its extensions [97] have provide contributions to this research area, but have been limited their scope to single domains. Active networks [122] had also been proposed as an extreme solution to the problem, allowing packets to carry instructions interpreted by the network fabric. Multi-domain approaches have mostly focused on single specific issues, such as anycast delivery or path selection to distributed services [95, 96]. Similar to ASR, Internet standardization organizations have also introduce overlay approaches for custom routing [102]. ASR in *NOVN* differs from previous work by providing a distributed and integrated solution for deploying both advanced network control and allowing applications to influence network layer decisions. Lastly, *NOVN*, through the employed named-object abstraction, belongs to the categories of Information Centric Networking [123, 124, 16] and name separation [100, 101] works.

3.9 Summary

This chapter presents *NOVN*, a novel network virtualization architecture aimed at providing a clean and logically simple solution for deploying virtual networks. Exploiting the named-object abstraction, *NOVN* provides a solution that offers the logical simplicity of L2 network virtualization which augmented with the advanced mobile edge cloud (MEC) techniques such as application specific routing, network slicing and QoS control, achieves a high degree of flexibility in creating customized topologies and routing of traffic in an application-aware manner. Results based on a working prototype deployed on the ORBIT testbed demonstrate that the new framework provides an efficient realization for defining and managing virtual networks without compromising performance or incurring excessive control overhead. Performance evaluation of various MEC scenarios are presented and the solution is compared with the overlay based virtual networks. Results show that *NOVN* provides faster path recovery and incurs no packet loss during link failure. The ASR improves the latency performance by 30% as compared to the baseline approach for a 90 percentile response time at 68 ms.

Chapter 4

Service Migration in MEC

Achieving low-latency services in the edge clouds face challenges of resource assignment and load balancing due to variability of user location (mobility), server load and network state. Dynamic resource migration techniques are considered necessary to achieve load balance, fault tolerance and system maintenance objectives. Container migration is emerging as a potential solution that enables dynamic resource migration in virtualized networks and mobile edge cloud (MEC) systems. This chapter proposes a traffic aware container migration approach and validates it with an end-to-end system implementation using a pure container hypervisor called LXD (Linux Container Hypervisor). The container migration model is then evaluated for real-time applications such as license plate recognition running in a mobile edge cloud scenario based on city-scale mobility traces from taxicabs in San Francisco. The system evaluation considers key metrics associated with application quality-of-experience (QoE) and network efficiency such as the average system response time and the migration cost for different combinations of load, compute resources, inter-edge cloud bandwidth, network and user latency. A specific compute resource and network-aware distributed resource migration algorithm called "ShareOn" is proposed and compared with alternative techniques using the San Francisco MEC model.

4.1 Introduction

Emerging cloud-assisted mobile applications – Augmented/Virtual Reality (AR/VR) involve intensive computation with real-time response constraints. Mobile edge computing (MEC)[41, 82] is currently under active consideration as a promising approach which supports low-latency applications by bringing compute, network and storage close

to the user. Edge cloud is generally deployed with limited compute resources to target local users. MEC is a distributed computing infrastructure that must respond to factors such as user mobility, fluctuating load, variability in network performance/congestion, and resource heterogeneity. Thus, maintaining user quality-of-experience (QoE) via distributed coordination between local edge cloud clusters is a challenge.

User QoE can be maintained in the MECs using a number of distinct mechanisms[74, 125, 126]. These include the use of either centralized or distributed resource assignment schemes which allocate computing servers to mobile user requests at nearby servers with available compute capacity. Resource virtualization (VM's and virtual networks) can also be used to partition and control resource use between multiple competing applications or users [127]. It is also possible to employ GPU's and/or parallelize computing resources across the network to accelerate the computation[128] or predict the network traffic [129]. Further, mobile user performance can be dynamically optimized via container migration in which the cloud process is moved from one computing node to another in response to mobility events and to load balance across the network. Container migration to be addressed in this work implies moving a virtual machine (VM) or a container from one edge cloud to the other[130]. Virtualization based on containers allows users to run an application and its dependencies in an Operating System with flexible resource allocation, easy scaling and improved efficiency[131, 132]. Containers are gaining momentum due to their light running and deployment overhead, smaller start and stop time, size, and higher network bandwidth as compared to VMs[133, 134].

Container (LXD)[135]/Docker[136]) and VM migration are implemented in [137, 138]. In [139, 140], migration algorithms have been designed based on a limited set of parameters e.g. distance between edge cloud and user. Existing literature either studies VM or implements container migration without explicitly taking the container specific parameters e.g. dynamic resource allocation (available processing speed, RAM and bandwidth) and size, into account. The migration cost of a heterogeneous system is a complex combination of local, remote and network resources. System load, available processing resource and inter-node bandwidth affect the total migration time. Furthermore, considering above mentioned parameters to simulate container migration to test

feasibility in a city-scale scenario is still unexplored.

This work aims to develop a more general approach to container migration and to validate the proposed methods via simulation of a realistic MEC scenario. Thus, we propose ShareOn*, a traffic-aware container migration algorithm using LXD and CRIU (Checkpoint Restore in Userspace)[141]. An end-to-end migration framework running real-time application has been deployed to analyze the impact of resource allocation, latency, bandwidth, size and migration time. A simulation model is also set up in which the containers are hosted in an edge cloud network running an automated license plate recognition (alpr [142]) application. Real traces from taxicabs in San Francisco [143] are used to model user mobility. Scalability of the system with respect to increasing traffic load is investigated using the above mentioned city-scale MEC model.

The rest of the chapter is organized as follows. Section 4.2 highlights the need of container migration in MEC. Section 4.3 describes the container migration system with specific details on the migration flow and the emulation set-up. Section 4.4 introduces the simulation model and its parameters. Simulation scenarios and analytics are detailed in Section 4.5. Results are discussed in Section 4.6. Section 4.7 provides a use case of container migration in the advanced driver assistance system (ADAS), and Section 4.8 provides a summary of the chapter.

4.2 MEC and Container Migration

The architecture of MEC is based on three layers of computing, the first at the mobile client, the second at a local network attached edge cloud, and the third at a centralized data center/cloud in the core of the network. As shown in Fig. 4.1, this architecture offers the advantage of low latency response to real-time applications which cannot tolerate a typical edge-to-core round trip delay that typically exceeds ∼100 ms.

With the help of flexible resource provisioning and sharing among neighboring edge cloud nodes, MEC can meet the unpredictable traffic demands and quickly scale the network due to its multitenancy feature. Further, in order to reduce the application

*The work on ShareOn was done in collaboration with Shalini Choudhury.

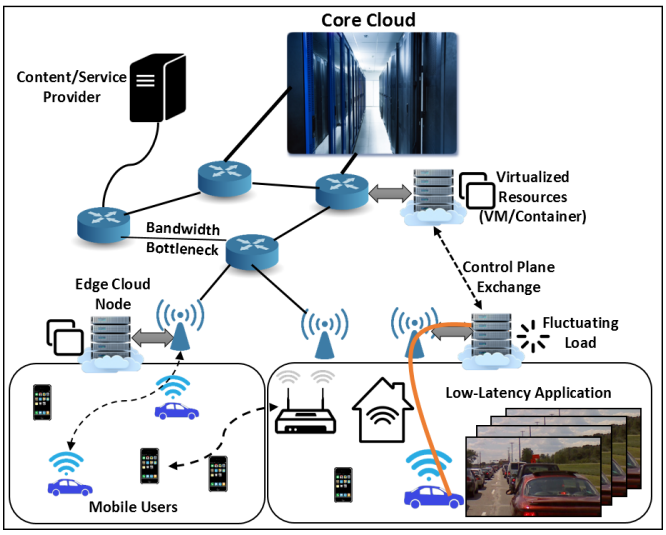

Figure 4.1: General MEC System

latency and to provide the required user QoE, service requests are handled by the resource virtualized environment. Container-based virtualization is finding increasing adoption in MEC systems to realize slice isolation and fine resource control. Resource isolation (especially, memory) across components of different applications is necessary for the integrity of individual applications. The light-weight containerized resources can be shared with neighboring nodes using migration techniques[144]. The dynamic container migration approach therefore can be used to address deteriorating user QoE, arising from the processing latency (system load) and/or the network latency (user mobility). In the rest of this work, we describe and validate a container migration system suitable for MEC scenarios with latency constrained applications.

4.3 Container Migration System

4.3.1 Flow Diagram

We start with an outline of the software stack at each MEC computing node. The migration process has two entities, a source, which is the host that initially has the container, and a sink, the container receiver. The prepare phase is succeeded by iterative pre-copy, freeze, state-copy, unfreeze and post-copy phases. Fig. 4.2 illustrates the

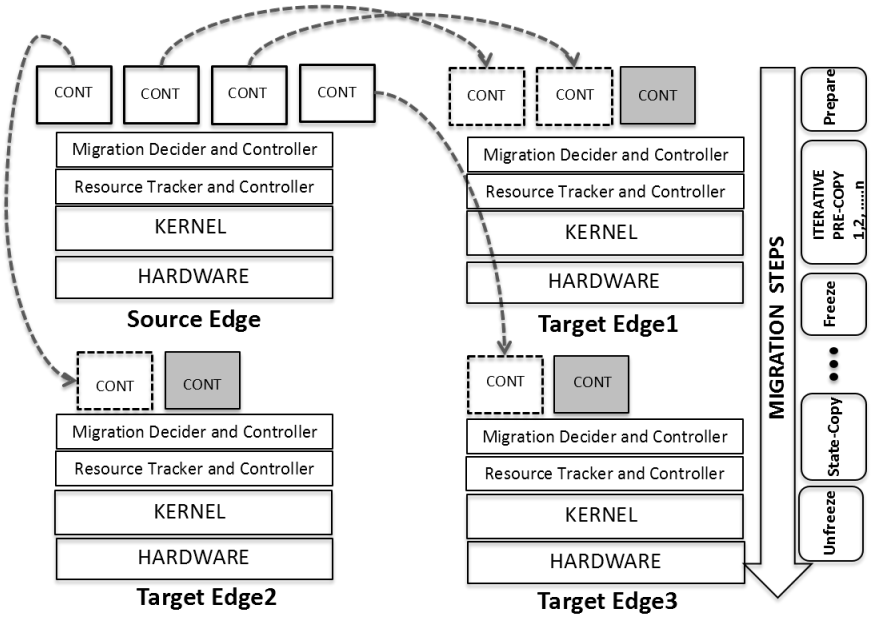

Figure 4.2: Container Migration Flow Diagram

distributed process flow by placing control and decision logic at each of the node. The resource tracker and controller at the source node assess the network and neighboring node resources. The migration decider determines best destination node while the controller initiates the selected container migration.

4.3.2 System Details

Our set-up is deployed at the SB9 (sandbox9) in the ORBIT[86] testbed (Fig. 4.3) which enables a software based emulation. We use SSH tunneling to connect to the edge cloud nodes. The core of LXD is a privileged daemon, which provides a REST API over a local UNIX socket and the network. Container orchestrated with Shell and Python scripts, allows us to run alpr remotely and to emulate users. CRIU does container checkpoint and restore on the host as a snapshot.

Container Creation: The primary sockets used in container migration are: control stream, CRIU images/mig/ stream and filesystem stream. LXD supports creating/managing bridges, IPv4 address, NAT (Network Address Translation) and DHCP (Dynamic Host Configuration Protocol) range. Hence, before setting up the container we set-up LXD

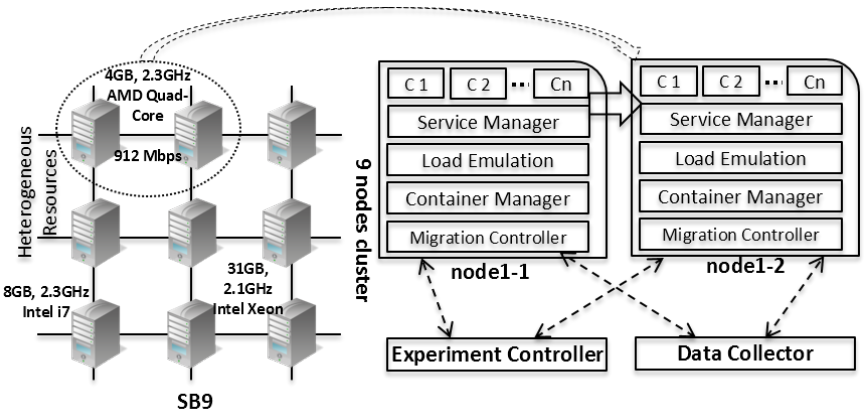

Figure 4.3: Container Migration Set-up in ORBIT

for storage and networking needs e.g. daemon settings, storage pools, network devices and profiles. Hostname is pre-added to the LXD group and the LXD tools are pre-installed to the destination edge with valid keys.

Migration Phases: There are three phases of migration: decision, initiation and completion. During the decision phase, a node accumulates resource, network, and application QoS information. A low overhead control plane protocol is used to exchange both routing and computing state information between edge cloud clusters in the region. Upon selecting the container to be migrated, neighboring edge clouds are queried using an extended inter-domain protocol such as EIR (edge-aware interdomain routing)[145] which has the above mentioned features. The neighboring nodes then respond with their utilization level: high, medium or low. A suitable destination node is chosen and the migration is initiated. Container state is then copied and traffic switched. On migration completion, the source node discards the old container states.

Application Details: The alpr is used to detect license plates of cars. The frames are obtained from the UE (User Equipment) video stream. After processing these frames in a container, the output is possible plate numbers with the set confidence level. The application phases include detection, binarization, char analysis, plate edges, character segmentation, OCR (Optical Character Recognition) and post processing.

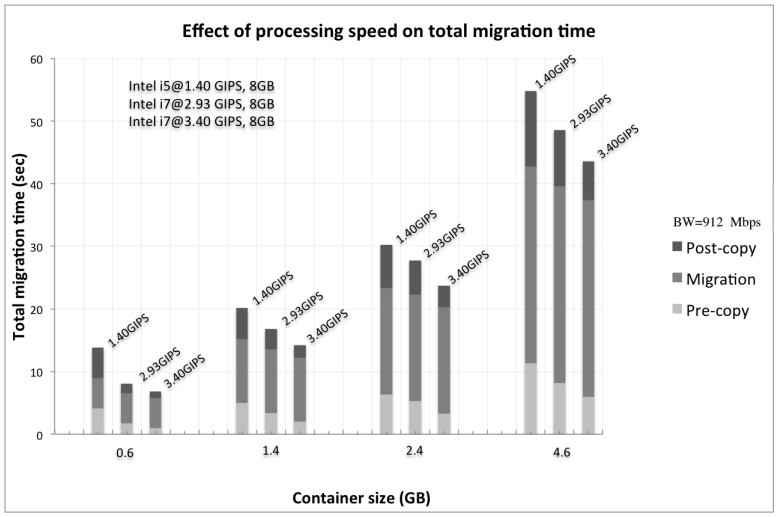

Figure 4.4: Impact of Processing Speed, Container Size on Total Migration Time

4.3.3 Assessing Migration Cost

We developed an experiment to measure migration cost parameters: pre-copy, migration, and post-copy time, with respect to machine type, network bandwidth and container size for alpr. In order to assess migration requirement and cost, different size containers (0.6-4.6GB) are exchanged between two similar test nodes by varying the processing speed. Fig. 4.4 shows the total migration time (tmt) which includes pre-copy, migration and post-copy time. The migration time for fixed sized containers and given inter-edge bandwidth remains same. The pre-copy and post-copy time are inversely proportional to the processing speed since it is also shared by the containers for the application specific computations.

4.4 Modeling Container Migration

We escalate the emulation set-up to carry out a large scale simulation model for a continuous container migration approach. To evaluate the efficiency of our work we choose San Francisco city as the geographical location with nine nodes spaced out across the city to deploy the edge cloud network as shown in Fig. 4.5. Real SFO taxicab traces are used from the heavy traffic routes across this location.

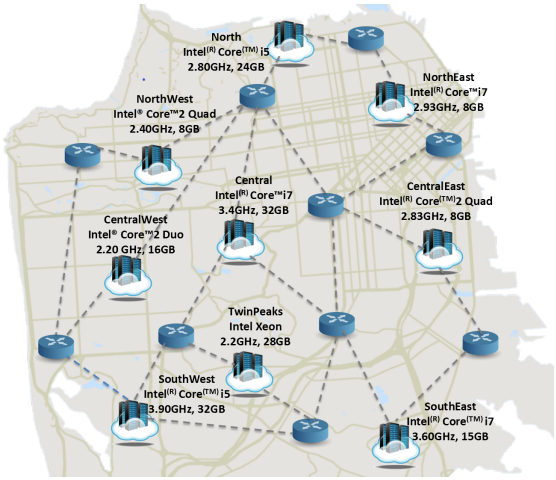

Figure 4.5: Edge Cloud Network Topology

4.4.1 Simulation Parameters

The parameters used in simulation are described below.

- Page dirty rate (r_{pd}): Memory pages modified per second
- Page size (s_{page}): Default size of a memory size
- Processing speed (s_p): Processor speed in GIPS
- RAM (m): Random access memory in GB
- Inter-edge bandwidth (b_i): BW between two edge nodes
- Network latency (t_n): User to edge cloud network delay
- Queuing latency (t_q): The wait time of request at an edge cloud node
- Processing latency (t_p): Computation delay at a node
- Control latency (t_c): Control plane delay between nodes
- Total response time (t_{total}): Sum of t_n, t_q and t_p
- Edge cloud load ($load$): Current load at an edge node
- Container size (s_{con})

- Containers-up (k): No. of running containers at a node
- Total migration time (tmt): Sum of pre-copy, migration and post-copy time

4.4.2 Migration Cost

For a given container to be migrated, the cost is a function of pre-copy time, post-copy time and migration time. Therefore, the total migration cost is defined as follows:

$$C_m = \sum_{j=1}^{N} \left[\frac{(k_{in,j} + k_{out,j}) * r_{pd,j} * s_{page,j}}{s_{p,j} * (1 - load_j)} \right] + \sum_{j=1}^{N} \sum_{l=1(l \neq j)}^{N} \left[\frac{k_{in,j,l} * r_{pd,j} * s_{page,j}}{b_{i,j,l}} \right] \quad (4.1)$$

In the Eq. 4.1, the first part denotes the compute time and the second part estimates the migration time due to inter-edge bandwidth, where N is the number of edge cloud nodes, and k_{in} and k_{out} are the number of incoming and outgoing containers from the given node respectively. The objective is to minimize system migration cost as well as average latency (application). In a continuous migration process, multiple container migration time overlaps which can be optimized by minimizing the time of maximum cost migration as follows:

$$\min . \left[\max . \left\{ \frac{(k_{in,j} + k_{out,j}) * s_{page,j} * r_{pd,j}}{s_{p,j} * (1 - load_j)} + \sum_{l=1(l \neq j)}^{N} \frac{k_{in,j,l} * s_{page,j} * r_{pd,j}}{b_{i,j,l}} \right\} \right], \forall j \in [1, N] \quad (4.2)$$

4.4.3 ShareOn: Migration Decision Algorithm

Migration is a two way process. First, the source node selects containers based upon their resource usage and application performance. Second, the source node determines *best* available destination node based upon its utilization region, estimated application performance and the available inter-edge bandwidth. The utilization region, *util*, captures the node specific parameters such as load, available CPU and RAM. Estimating application performance at the destination node is challenging because of network variability, fluctuating node utilization and the application type. Also, the number of

Algorithm 1 Container Pre-selection for Migration

Input: k, t_{th}, t_{total}
Output: Pre-selected containers

1: **for** cont ∈ EdgeNode **do**
2: **if** $(t_{total} > t_{th})$ **then**
3: $cont \rightarrow List_{pre-select}$
4: **end if**
5: **end for**
6: **SORT(descending)** $\text{List}_{pre-select}(\text{t}_{total})$

Algorithm 2 : Utilization Regions (UR) of a Node

Input: $s_{p,avg}, m_{avg}, k, s_{p,th1}, s_{p,th2}, m, m_{th1}, m_{th2}$
Output: UR(L, M or H)

1: **Calculation**
2: $s_{p,avg} \leftarrow s_p/k$; /*node calculates average processing*/
3: $m_{avg} \leftarrow m/k$; /*node calculates average memory*/
4: **if** $s_{p,avg} \leq s_{p,th1} || m_{avg} \leq m_{th1}$ **then**
5: $H \rightarrow UR$
6: **else**
7: **if** $(s_{p,avg} > s_{p,th1} || m_{avg} > m_{th1})$ AND $(s_{p,avg} \leq s_{p,th2} || m_{avg} \leq m_{th2})$ **then**
8: $M \rightarrow UR$
9: **end if**
10: **else**
11: $L \rightarrow UR$
12: **end if**
13: **STORE** UR

migrations between target nodes affect the available inter-edge bandwidth and therefore, the migration decision must take this into account.

ShareOn works as follows. First, a node shortlists high total application latency containers and determines the primary reason — high processing latency, networking latency or both. Second, for each container, a few suitable neighbors are selected using two conditions: (a) falls in low or med *Util* regions and/or (b) geographically closer to the user. Algorithm 1 pre-selects the containers to be migrated based upon their application latency threshold.

The average processing per container at a node is $s_{p,avg} = s_p/k$ and the average RAM is $m_{avg} = m/k$. The utilization region of a node is found using algorithm 2 which defines High, Med or Low zones by capturing available processing speed and RAM, and comparing them with pre-defined values needed to run an application.

The network latency, t_m, between the user and the destination edge cloud node can be estimated as $d_{u,e} * l_d + var(l)$ where $d_{u,e}$ is the distance between the user and the destination edge node, l_d is the network latency per unit distance and $var(l)$ is the past moving average latency variance of the user geographic area and the destination node. The processing latency, t_p, of a destination node can be approximated as $t_{p,avg} + utilFac * \alpha_e$ where $utilFac$ is the current utilization factor of a node, and α_e is the latency factor associated with the current utilization.

Migration gives rise to the computation and network overheads. Therefore, the decision algorithm has to adequately determine whether, when and where to migrate depending on aspects such as application QoS, user mobility, inter-edge bandwidth, and resource availability at MECs. Considering these parameters, we design ShareOn, a dynamic container migration technique which uses traffic (number of requests per second at a node) as a primary metric. Each node tracks its own compute and networking resources, and is aware of the application latency of each request. These nodes have control plane connectivity to each other and hence can query the neighboring nodes periodically for the above mentioned parameters. Thus, in our system, each node can decide, control and migrate its containers in a distributed manner. Finally, a destination is chosen based upon number of migrations from the source to the destination considering the available inter-edge bandwidth and compute. The complete procedure is described in the algorithm 3.

4.4.4 Result — Emulation

The emulation based total application delay is shown in Fig. 4.6 to observe the effect of server load on the container migration and the application performance. ShareOn is compared with a case when there is no migration. The load is varied from 0 to 1 for node1 and 1 to 0 for node2 in steps. When the load reaches 0.4 for the node1, the total application delay crosses a set threshold (100ms in this case), triggering migration. Node2 is chosen using ShareOn which shows a drop in the total delay upon migration completion.

Algorithm 3 Final Containers and their Destination Selection for Migration

Input: $List_{preselect}, C_m, L_{est}$

Output: Final selected containers for migration and their destination edge cloud nodes

for cont $\in List_{preselect}$ **do**
  for DestNode $\in EdgeNodes$ **do**
    if $DestNode \in UR\{L, M\}$ AND $t_{total} - t_{est,dest} > \delta L$ **then**
      $DestNode \rightarrow List_{dest}$
      $cont \rightarrow List_{select}$
    end if
  end for
end for
for $cont \in List_{select}$ **do**
  $FinalDest = min(C_m) \forall List_{dest}$
  $FinalDest \rightarrow$ List$_{cont,dest}$
end for

4.5 Simulation Set-up

We develop a simulation set-up similar to the emulation model described in the Section 4.3, to test the scalability in a large geographical region such as San Francisco city with real mobility traces. The topology is same as shown in Fig. 4.5. This section details the simulation scenario and the numerical values for the parameters described earlier.

4.5.1 Simulation Scenario

Initially, we determine the system performance without considering migration, using following approaches.

Equal-Load (E): The user requests are equally divided among the heterogeneous edge cloud nodes, routing a set number of requests based upon user vicinity. The remaining requests are routed to the next closest node and so on.

Nearest-Edge (N): The user requests are always routed to the closest node irrespective of the node's current load.

Migration is simulated using the approaches listed below.

Bandwidth-only: The users are connected at the nearest edge cloud node and the migration is done based on the application QoS and available inter-edge cloud bandwidth.

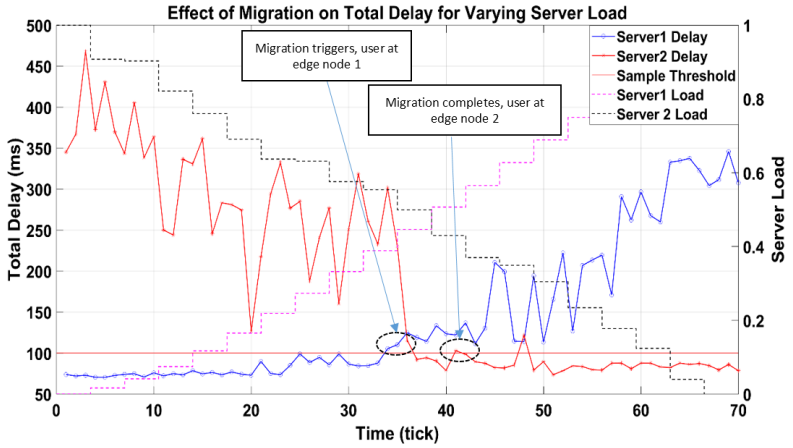

Figure 4.6: Baseline (no migration) vs. ShareOn

Processing-only: Following the above process, the migration is done based on the application QoS and the destination node resources (processing speed).

In all the cases, user mobility is introduced and load is varied by injecting multiple requests per taxicab. The delay induced by the control latency, t_c, is negligible and is omitted in the simulations. Our proposed approach, *ShareOn*, can be instantiated from the nearest or the equal load scenario considering their first connected request as the initial state.

4.5.2 Simulation Parameters

The numerical values for the simulation are listed in Table 4.2. There are total 536 taxicabs in the city with a known mobility pattern. The load is varied from 0 to 1 by initiating multiple requests from a taxi. The processing speed determines the page dirty rate which allows us to push most of the memory pages to destination before suspending the container at the source.

4.6 Results and Discussion

This section presents the results obtained from the simulation model introduced in the previous section.

Table 4.1: Simulation Parameters

Parameter	**Value**	**Parameter**	**Value**
s_p	2.2-3.9 GIPS	b_i	10-100 Gbps
r_{pd}	2.5-4 kpps	s_{page}	4-64 KB
m	8-32GB	$load$	0-1
s_{con}	0.6-4.6 GB	#taxi	536

4.6.1 System Performance

Fig. 4.7 compares our migration approach, ShareOn with no migration cases. In the equal load case, the average system response time at load 0.1 is low as compared with the other approaches since the service requests have been equally distributed among nine edge cloud nodes. However, as load gradually increases, the average system response time starts degrading since without any optimization this approach cannot handle the volume of requests with the available resources at each node. While in the case of the nearest edge (users connect to the closest MEC) the average system response time is prominent. The reason for this rise is that in the real-time mobile taxicab trace, many of the users connect to the North-East edge cloud node (ref: Fig. 4.5) because of their close physical proximity to that node when they are introduced into the system. The limited resources at that node are unable to support all the connected users and hence the system enters into overload resulting in a large average system latency.

Using ShareOn-E, initially there are not many migrations since all the requests are well distributed between nodes. On increasing the load, the system response is well below the non-migration approaches, as the load gets efficiently distributed across the geographical regions. This is due to ShareOn taking processing capability, inter-edge bandwidth and network latency of each node into consideration for migrations. In the case of ShareOn-N, the users are initially connected at the nearest edge cloud where the available resources are exhausted resulting in a slight increase in the migration cost.

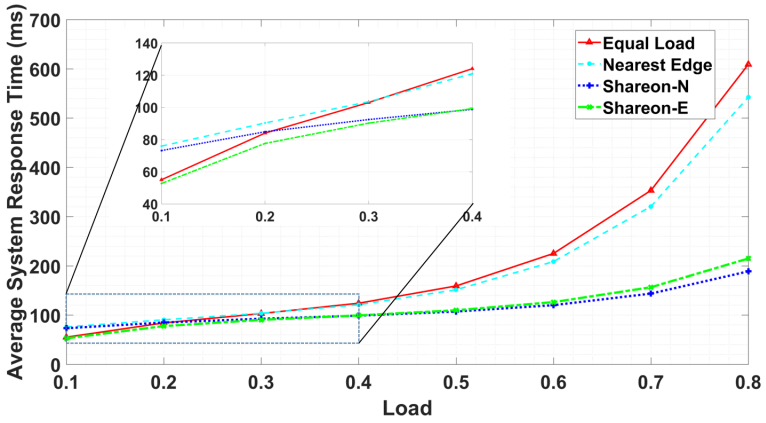

Figure 4.7: Average System Response Comparison for Static Allocations and Migrations

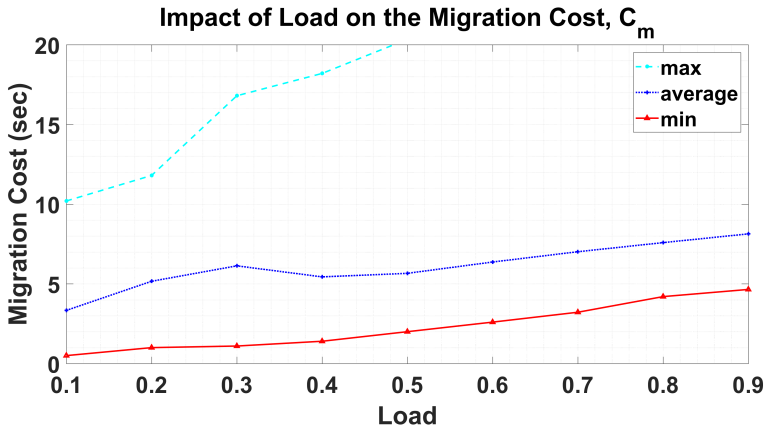

Figure 4.8: Effect of Load on the Migration Cost

4.6.2 Migration Cost

Fig. 4.8 presents the migration cost (C_m) for different system load. The maximum C_m is the peak migration time recorded for the system at a given load. Similarly, minimum C_m is the least while the average C_m is mean time. The total number of containers migrated are not linearly proportional to the load. At a higher load, some containers are omitted even though the latency threshold is met for the migration due to scarce compute and networking resources. Therefore, even though the number of migrating containers rise with the load, a drop in the average migration cost can be observed, as seen at load=0.3.

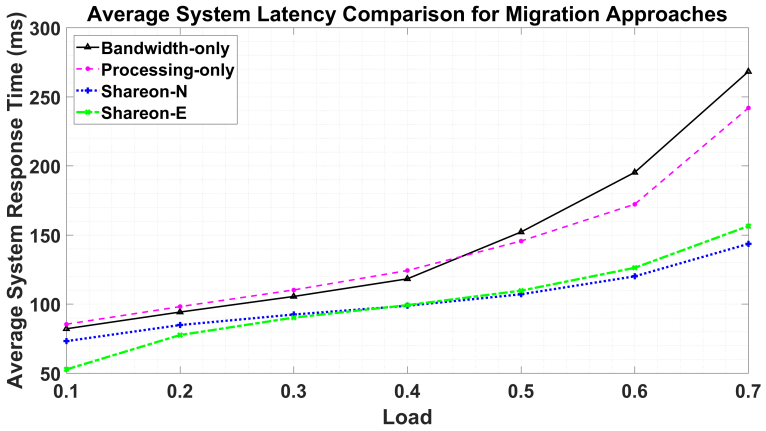

Figure 4.9: Average System Response Comparison for Different Migration Approaches

4.6.3 ShareOn vs. Other Approaches

Fig. 4.9 compares ShareOn with bandwidth-only and processing-only migration methods. The former performs better than the later at lower loads as the migration time dominates the pre-copy and post-copy time for the fewer container migrations in these cases. As the bandwidth-only approach keeps track of bandwidth before initiating migration, the average system latency does not suffer from the migration time factor. For the higher load scenario, checking bandwidth only is not sufficient as the migration time dominates the pre-copy and post-copy time, thereby increasing the total system latency. In either case, ShareOn performs significantly better than these methods. The project code is available at [146].

4.7 Supporting Autonomous Driving

As a use-case of container migration described in this chapter and named-object based virtual networks detailed in the previous chapter, in this section, we present EdgeDrive which is a networked edge cloud service framework to support low-latency applications during mobility taking into account needs of the driver, nature of the required service and key network features. We implement head-mounted device (HMD) based Augmented Reality (AR) ADAS applications such as navigation, weather notification and

annotation based assistance to drive the evaluation. These services are then coupled with the Mobile Edge Clouds (MECs) wherein the container based service migration is enabled based upon migration cost and required Quality of Experience (QoE) to support mobility. An emulator based evaluation is carried out on the ORBIT testbed using realistic San Francisco taxicab traces running over nine edge cloud nodes and AR HMD being used by drivers. The experiments show that the EdgeDrive can support low-latency ADAS applications with an average system latency less than 100 ms for the applications under consideration.

4.7.1 ADAS

Advanced Driver Assistance Systems (ADAS) are expected to become increasingly important to the automotive industry [147]. ADAS focuses on assisting drivers by providing timely critical information such as real-time navigation, safe driving limits, pedestrian crossing, and sensor calibration. Recent advances in ADAS focus on applications such as 3D mapping [148], Internet of Vehicles (IoV) [149], and holographic displays [150] requiring uninterrupted information exchange between connected cars, offloading computation to the cloud computing servers, and handling mobility through novel communication as well as networking architectures [127, 16].

Head-mounted device (HMD) based Augmented Reality (AR) provides a user-friendly method for drivers of vehicles to interact with the surrounding environment by supplementing real world with the contextual information, e.g., traffic status, stop notification [151]. Furthermore, gesture enabled head-mounted AR devices make this interaction hands-free thus improving the user Quality of Experience (QoE). Nevertheless, the core functions of AR enabled driving assistant applications (perception, annotation, visualization and sensing) are usually computation and data intensive [152], and the on-board computing capabilities of currently available AR devices such as Microsoft Hololens [43] and Google glass [44] are insufficient to perform these intensive tasks. For instance, the latest Hololens has only 1.04 GHz CPU clock rate, and 2GB RAM. The constrained capability cannot satisfy the service demands associated with processing analytics over a single frame within 30 ms and delivering a 60 frame per second [153].

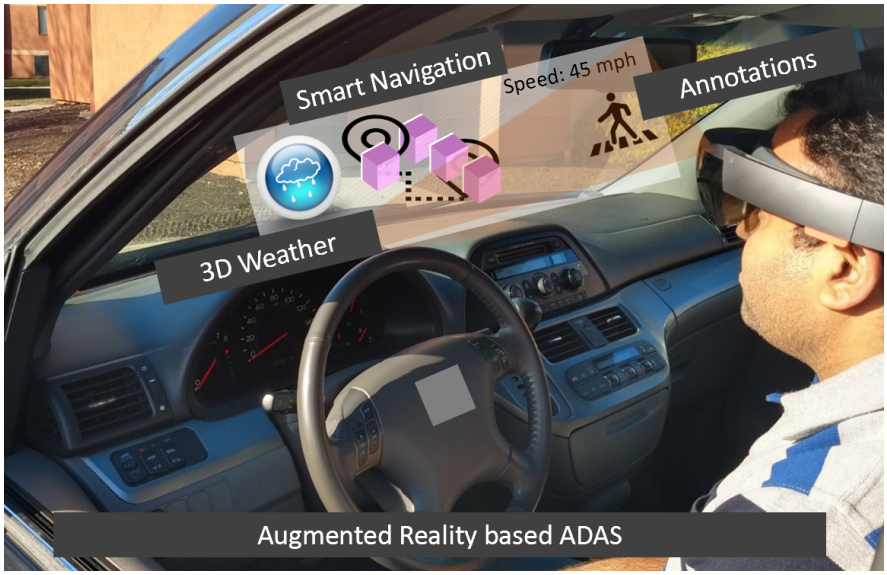

Figure 4.10: AR-enable ADAS Applications. The dashboard displays the weather, navigation and surrounding information.

Further, the applications running on these devices rely on shared information obtained from nearby vehicles or roadway infrastructure to interact with the environment. For example, self-driving vehicles receive surrounding information from other vehicles which have knowledge about distant traffic for better informing driving decisions. This inter-vehicle information exchange can be realized either as a peer-to-peer (V2V) application or as a cloud service based on edge cloud infrastructure. As central cloud servers may be at distant geographic location from the source of data generation, conventional offloading is likely to introduce significant network latency. For a client instance in New Jersey which connects to Amazon EC2 cloud servers located in West Virginia, Oregon and California, the round-trip latency *alone* is 17, 104 and 112ms, with achievable bandwidths of 50, 18 and 16 Mbps, respectively.

Mobile Edge clouds (MECs) bring computation, storage and networking close to the user thereby promising to support stringent latency requirements for AR applications [81, 83, 154]. Latency is a critical factor in user QoE for AR applications. For example, in an HMD, the combined network and processing latency while using an

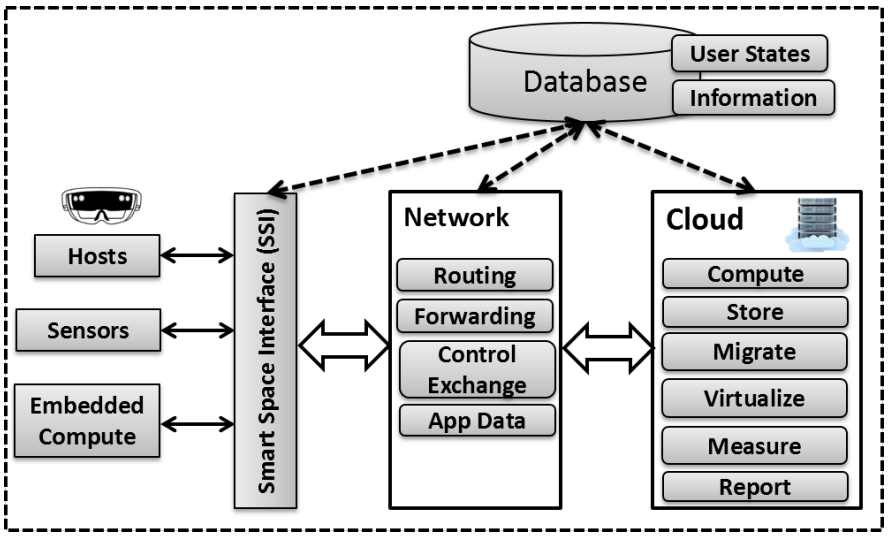

Figure 4.11: ADAS System Design

AWS cloud is more than 100 ms at which 50 degrees per second head or object rotation introduces 5 degrees of angular error [155]. In case of objects closer to the user, this implies that the user views a virtual coffee cup in the air instead of on the table and for farther objects, the error only accumulates. MECs provide a way to reduce this latency, and hence the overall errors by providing responses faster than the user perceivable latency.

EdgeDrive is an edge cloud based end-to-end system to minimize the latency of AR applications for ADAS. We implement and deploy several example ADAS based AR applications such as smart navigation, weather notification, and annotation based assistance as shown in Figure 4.10. The applications are augmented with networking and edge cloud components emulated using the ORBIT [156] testbed with demonstrated methods to reduce overall system latency by edge cloud techniques such as caching, application specific routing, dynamic server selection and edge cloud migration [75]. A large-scale emulation using nine edge cloud locations in San Francisco is used along with realistic taxicab traces to showcase dynamic server migration. It is shown that the MEC system can improve AR based ADAS performance when augmented with additional capabilities such as service containerization, and its migration.

The rest of this section is organized as follows. First, the above mentioned ADAS applications are described while detailing smart navigation application for the techniques used to reduce its delay. Next, the EdgeDrive architecture using MECs is provided with the emulation set-up used to carry out the experiments. Finally, results and discussion are presented to conclude the use-case.

4.7.2 ADAS Applications

In order to test the functionalities of ADAS system, we identified three key applications namely, (a) annotation based assistance, (b) smart navigation, and (c) 3D weather. These applications are chosen as they cover different features and requirements such as the latency, caching, throughput and compute as listed in Table 1. In particular, the first application supports the driver by embedding the processed surrounding information onto the AR device using image processing. The second application provides navigation support by embedding 3D objects onto the path where the driver collects the objects similar to a game play. The third application projects the current location's and destination's weather information on to the driver's AR display.

Table 4.2: Requirements for Sample ADAS Applications

Feature	**Annotation**	**Smart Navigation**	**3D Weather**
Latency	Low	Medium	High
Database/Caching	No	Yes	Yes
Throughput	High	Medium	Low
Compute	High	Medium	Low

4.7.3 ADAS System Design

The system consists of devices (e.g. Hololens), network (e.g. routers) and the service functions placed at each cloud servers. The server runs on an edge cloud which is typically one hop away from the Access Point (AP). The network, in addition to routing and forwarding, also supports control exchange functions among the edge clouds. The edge cloud provides features such as computation, storage, resource virtualization,

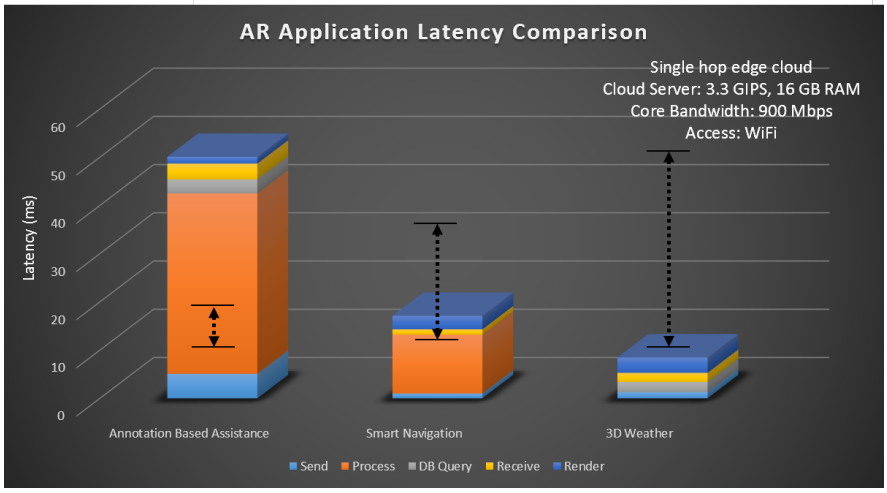

Figure 4.12: ADAS Application Latency Comparison

service migration, and performance monitoring. The applications are developed using Unity and C#. The design framework is as shown in Figure 4.11. The APIs developed at the smart space interface allows user to seamlessly access the services from the cloud server. The user states are accessible across the network using a logically centralized database.

By deploying this three-layered, device-router-edge cloud system on the ORBIT testbed, the function level latency is measured for all the applications as shown in Figure 4.12. The arrows represent the required latency thresholds which is low for annotation based application as the response should be in real-time as the user's head movement. The lower bound represents the latency which is sufficient to provide satisfied quality of experience to the user. The smart navigation application spends most of the time in image processing using OpenCV and therefore requires a highly computational, lightly loaded edge cloud for offloading the compute. The 3D weather application is able to serve without delays using its caching function retrieving the statistics from a central weather server periodically. It can be noted that despite edge cloud being one hop away, the annotation based assistance application is unable to achieve latency bounds and there require techniques such as distributed task computing [153] which are not studied in this work.

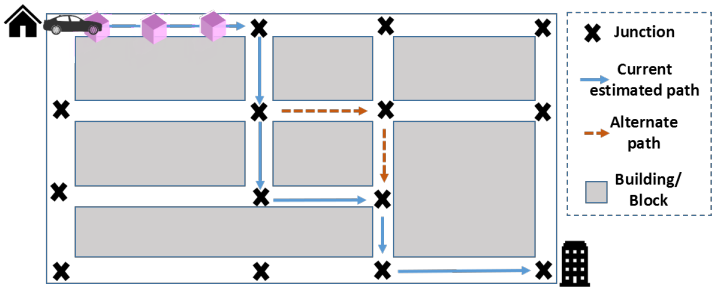

Figure 4.13: Smart Navigation in ADAS

4.7.4 Smart Navigation

Smart Navigation is one of the most crucial features in ADAS. First, we describe the challenges of developing an uninterrupted navigation service and then detail the suitable design choices.

Localization with Hololens: Navigation needs GPS location information of the device. Yet, currently available Hololens devices do not support the GPS module. Therefore, we rely upon mapping coordinates based upon accelerometer and gyroscope sensor readings for localization. The map is divided into set of junctions and at each junction, the route is recalculated based upon current Hololens coordinate and destination coordinates. Therefore, the complete navigation system relies upon manipulating coordinates as read by local sensors and thus the implementation functions without the GPS as shown in Figure 4.13.

High Uplink Bandwidth: Hololens requires to send the stream of images to the server continuously. This is expensive as it requires high uplink bandwidth. In order to optimize this, we send only when the image changes more than 5% (can be varied) as compared to the last sent image and thereby also saving server compute. A lightweight bitmap based hash function is locally employed on the HMD for the image comparison.

Path Rerouting: Sometime the vehicle may take paths other than that suggested by navigation. This is handled by continuously comparing the coordinates of line connecting two suggested junctions. If the user's current coordinates do not lie on the line, the path is rerouted to the next optimal path. Finally, the path is always selected based

upon the minimum distance between the source and the destination considering all the possible combinations.

Receiving Real-time Responses: The image processing and path calculations are offloaded to the neighboring edge cloud server and is detailed in the next section.

In a general ADAS system, a tagged metadata image stream is used for all the services rendered to an HMD. Therefore, in this work, we have employed image streaming irrespective of the application.

4.7.5 EdgeDrive System

In this section, the EdgeDrive system components and its capabilities are described.

System Components

The system has the following components as shown in Figure 4.14. (1) **End Devices**: These are hosts, sensors and embedded compute devices such as FGPA which can push/pull data from the system; (2) **Smart Space Interface**: It allows to connect heterogeneous devices to the edge cloud by mapping their API requirement to the deployed system; (3) **Network**: Its functions are routing, forwarding and control. The deployed network has additional feature which allows application state to be pushed to the cloud and vice-versa. This simple action allows application specific routing to best available edge cloud while keeping the control distributed in the network; (4) **Edge Cloud**: Each instance of edge cloud can compute, store, migrate resources by virtualization (such as containers), measure and report statistics to the network and neighbors; (5) **Database**: It contains user states and space specific information which can be cached at the edge cloud proactively or on-demand.

AR using HoloLens

HoloLens provides a state of the art technique to achieve AR functionalities by its unique features such as frame of references (stationary and attached), spatial anchoring and spatial mapping. This implies that a virtual object (hologram) can be placed and oriented at a fixed location in the real-world (mixed reality) and can be retrieved at

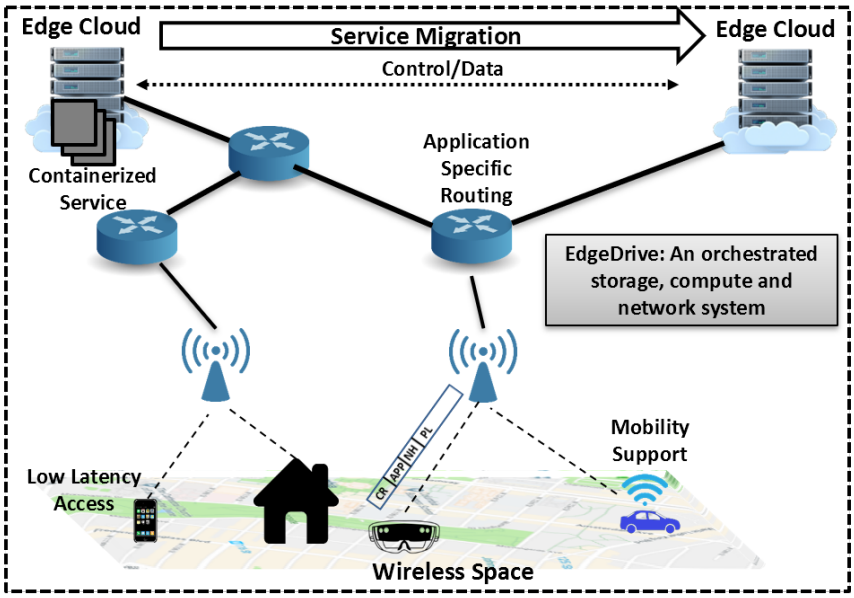

Figure 4.14: EdgeDrive System

the same location later anytime. The specially designed spatial coordinate system can be used to derive other coordinate systems or can be used as-is to determine device's own position in the three-dimension space.

EdgeDrive Capabilities

The key features of EdgeDrive are described as follows. **Service Containerization:** All the services such as navigation, annotation and weather reporting are containerized at the edge cloud servers. Containerization provides benefits such as service and user level isolation, faster start and stop, and easy migration.

Mobility Support using Container Migration: EdgeDrive provides ADAS application mobility support by migrating containers across edge cloud nodes. Each edge cloud node has a performance monitor, migration manager and controller which with the help of neighboring edge cloud resource information decides the target as shown in Figure 4.15.

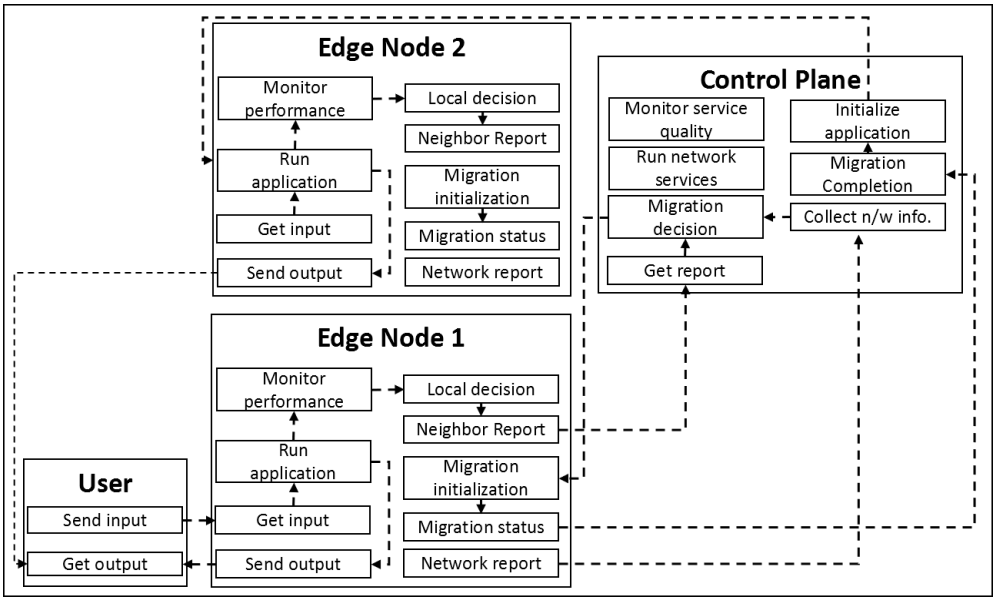

Figure 4.15: Container Migration in EdgeDrive

4.7.6 Experimental Details

The experiment is set up at the sandbox-9 (SB9) on the ORBIT [156] testbed. Nine nodes are set-up based on the topology of a San Francisco edge cloud network with variable inter-edge bandwidth (randomly chosen between 100-900 Mbps) connectivity as shown in Figure 4.5. The nodes are placed based on the population density in SFO and have heterogeneous memory, processing speed (Ref: Fig.4.5) and cpu load (varied from 0 to 1). Real SFO taxicab traces are used from the heavy traffic routes across the city. Each of these users are assumed to be using the smart navigation application as described earlier. The source and destination are chosen randomly. The low-latency requirement of the navigation application is aimed to be fulfilled by EdgeDrive by: (a) pre-caching and reevaluating navigation information at each junction and the line connecting the neighboring junctions, (b) associating user to the best available edge cloud, (c) providing service transfer support using container migration, and (d) enabling network embedded application specific routing.

4.7.7 Emulating ADAS

The ADAS applications are emulated on the ORBIT testbed by setting in-lab source and destination for navigation. The junctions are pre-defined at every intersection and realistic dimensions are obtained using a laser rangefinder. A navigation algorithm finds the shortest distance between the source and the destination. The coordinates are initialized at (x,y,z) = (0,0,0) using OpenCV based recognizable chilitags placed at the source. For the entire experiment, z (height) is set to 0. At source, the user looks at the marker using Hololens and the navigation begins for the set destination. The user then collects the pink cubes as shown in Figure 4.16(b) to navigate towards the destination. The image stream rate from HoloLens is varied from 10fps to 60fps for the walking user to emulate vehicular mobility. For a large scale experiment, 536 mobile users are injected into the system using a custom script emulating from the SFO taxicabs traces and thereby loading servers. Similarly, the annotation application is emulated by obtaining real-time printer information such as ink status as shown in Figure 4.16(c). MySQL database is used for storage and Apache for server code.

4.7.8 Container Migration

As the services run on containers at the nine edge clouds, the migration is self-triggered locally at an edge considering migration cost and the parameters as described in ShareOn. The migration has three stages: (a) decision, (b) initiation and (c) completion. During the decision phase, the right edge cloud is chosen for migrating a running containerized service. During the initiation phase, the pre-copy of container begins thus copying the pages from source to the destination. Finally, during the completion phase, source edge cloud node informs the destination edge cloud node to run the newly received container and itself discards the old container upon receiving confirmation from the destination. The complete process is automated using Python and shell scripts.

4.7.9 Results and Discussion

This section presents the results obtained using the experiments as detailed earlier.

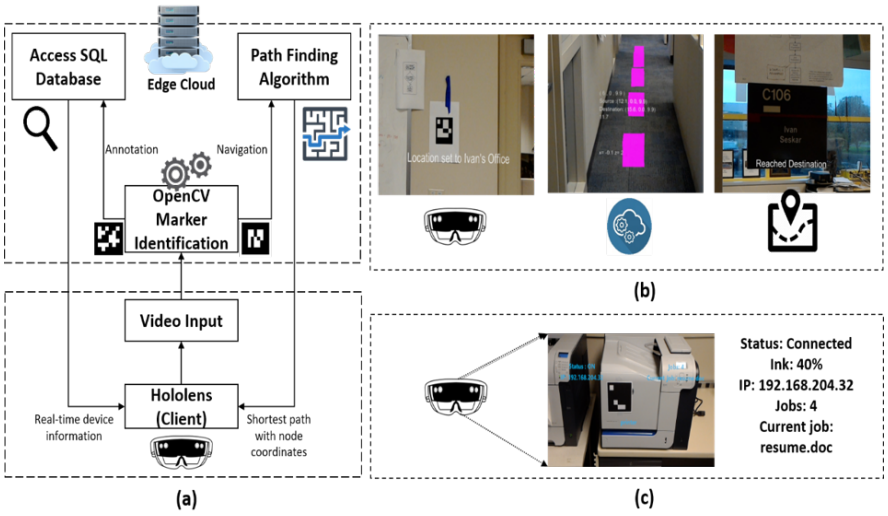

Figure 4.16: Emulating ADAS Applications at WINLAB

Parameters Impacting Container Migration

Container migration consists of pre-copy, migration and post-copy. For stateless applications such as annotation, weather and navigation, post-copy is not required and therefore merely migrating the dependencies enables the destination node to restart the services. Figure 4.17 shows the impact of machine type and container size on the total migration time. It can be seen that the migration time for a 4.2 GB container running on an Intel i5 machine can be as high as 60 seconds which is less useful for real-time applications described in this work. Therefore, while deciding service migration, along with the bandwidth, machine type plays an important role. The other parameters of interest are system utilization (load), processing capabilities of the edge cloud and the available RAM.

Factors Affecting Application Performance

The input from HoloLens in case of weather and smart navigation applications is primarily the coordinates (a few KBs) of user whereas in case of annotation application, continuous stream of images (avg. 30fps) is sent from the device to the server. Therefore, the annotation application is the most affected by the system load as shown in

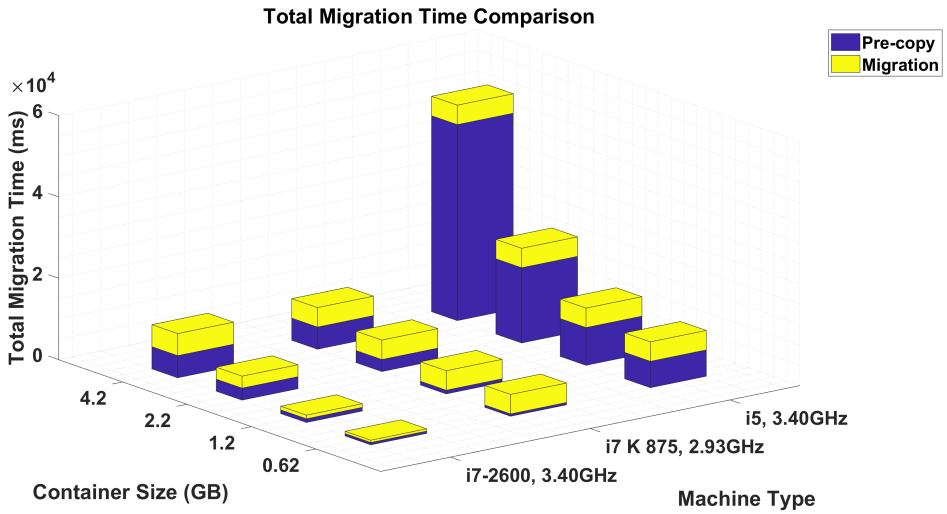

Figure 4.17: Impact of Machine type and Container Size on Total Migration Time (Bandwidth=912 Mbps)

Figure 4.18. As the weather application relies upon the pre-fetched data in the edge cloud, the current system load does not affect its performance. The navigation application intermittently requests server to calculate the path and therefore its round trip response time grows when the system load is high.

Figure 4.19(a) depicts that when the frame per second sent from the HoloLens to the edge cloud server is low, the server is unable to determine the object and therefore the response time is high. When the fps is increased, OpenCV is able to detect the marker or object and therefore the annotation appears in less than 70 ms. As the fps is increased, the server is loaded with the heavy processing and therefore the RTT increases again.

Figure 4.19(b) shows the impact of mobility on the navigation performance of HoloLens. The in-lab mobility is normalized from 0 to 1 and error is defined as junction missed due to mobility which otherwise would have provided the shortest path. The percentage error is calculated by running the experiment multiple times and then averaging the miss rate.

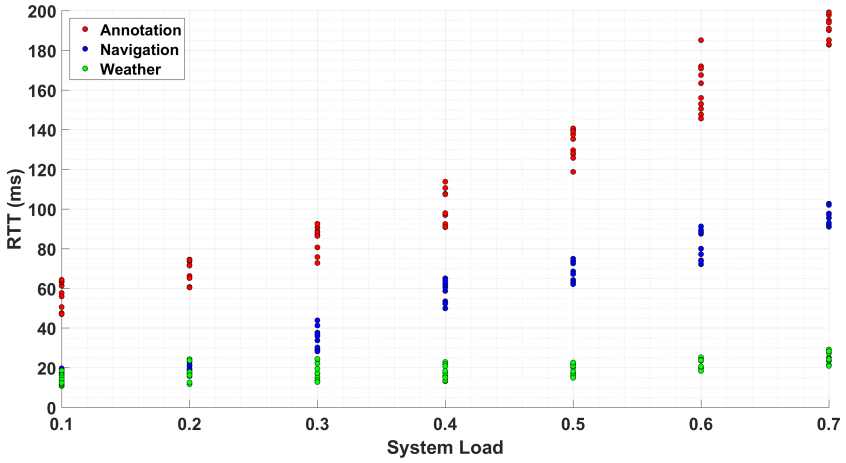

Figure 4.18: Impact of System Load on Application Performance

Effect of Container Migration

Figure 4.20(a) shows the latency for the navigation application for a single random user when the system is loaded at 0.5 (50% CPU utilization). As the user is mobile, the latency depends upon whether the user is moving closer to the assigned edge cloud or not. Employing ShareOn based EdgeDrive, the application latency for the user drops after the time tick 20 at which point the migration is complete and user's service is migrated to another edge cloud.

Figure 4.20(b) shows the average latency for the whole system for all the applications when the CPU load is 0.5. For the higher compute demanding applications, EdgeDrive is able to provide better QoE as the latency is significantly lower than the system without migration.

4.8 Summary

This chapter has proposed dynamic container migration as a mechanism for supporting user-mobility, server load and network fluctuations. A traffic-aware container migration method is developed using a testbed based set-up, to emulate an edge cloud network and user mobility. The migration cost is evaluated by running a real-time application and a distributed migration algorithm. Using parameters obtained from the emulation, a

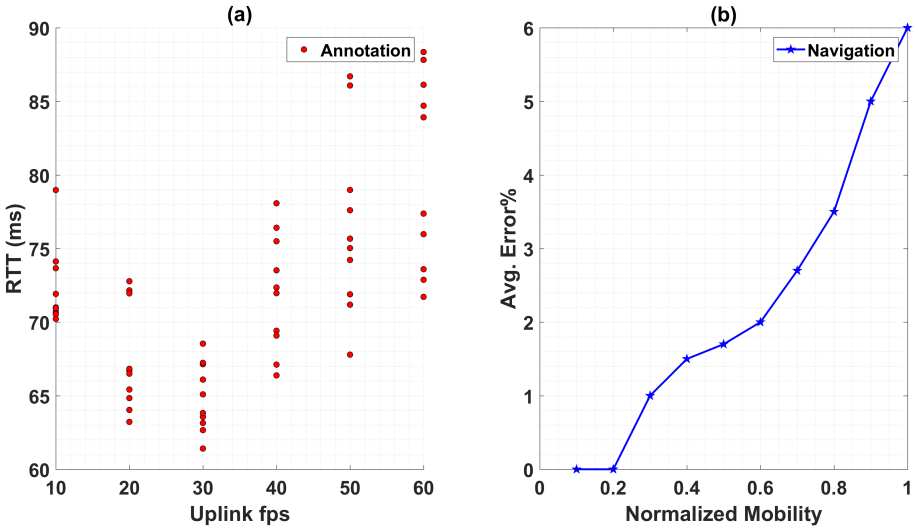

Figure 4.19: Impact of Various Parameters on the Application Performance. (a) Uplink fps affects the RTT for Annotation Application and (b) Mobility affects the Accuracy of Navigation

large-scale simulation model for migration is developed incorporating real taxicab traces from San Francisco city. A heuristic traffic-aware container migration scheme, ShareOn, is proposed which considers multiple parameters: application QoS, network latency, edge cloud resources, and inter-edge bandwidth. The system performance of ShareOn is compared with two non-migration approaches: equal-load and nearest-edge, and two migration-based approaches: bandwidth-only and processing-only. Furthermore, as a use-case we proposed EdgeDrive, a mobile edge cloud based compute, network and storage orchestrated architecture to support advanced driver assistance systems (ADAS). Using key Augmented Reality (AR) applications developed for the head mounted display (HMD), it is demonstrated that EdgeDrive is able to provide low-latency service to the driver during mobility. The key observations from this study are: (1) machine type plays a crucial role in deciding migration, (2) migration is a viable approach when sufficient computation (at source and destination) and inter-edge bandwidth are available, (3) a low-load scenario incurs substantial migration cost while there is no significant drop in the average system response time as compared to no-migration approaches, (4)

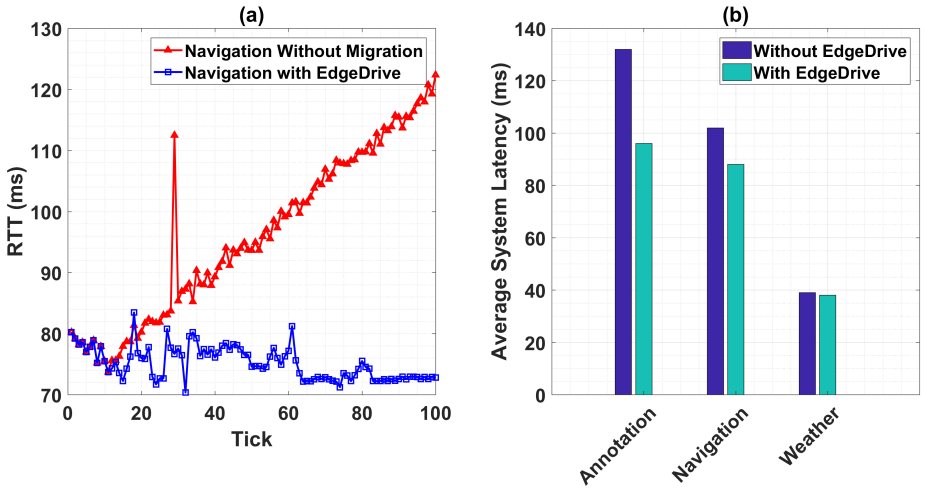

Figure 4.20: Latency Performance using EdgeDrive. (a) Single User Latency with and without migration and (b) Average System Performance with and without Migration

processing-only and bandwidth-only approaches fail to lower the average system response time at higher load as compared to the multi-parameter ShareOn approach, (5) applications requiring higher compute for instance annotation based assistance should be offloaded to the closest available edge cloud, (6) the latency of applications requiring pre-fetched data cannot be significantly optimized, and (7) service migration should consider network bandwidth, system load, and compute capability of the source and the destination.

Chapter 5

Distributed Control Plane Protocol for MEC

This Chapter presents a novel control plane protocol designed to enable cooperative resource sharing in heterogeneous edge cloud scenarios. While edge clouds offer the advantage of potentially low latency for time critical applications, computing load generated by mobile users at the network edge can be very bursty as compared with aggregated traffic served by a data center. This motivates the design of a shared control plane which enables dynamic resource sharing between edge clouds in a region. The proposed control plane is designed to exchange key compute and network parameters (such as CPU GIPS, % utilization and network bandwidth) needed for cooperation between heterogeneous edge clouds across network domains. The protocol thus enables sharing mechanisms such as dynamic resource assignment, compute offloading, load balancing multi-node orchestration and service migration. A specific distributed control plane (DISCO) based on overlay neighbor distribution with hop-count limit is described and evaluated in terms of control overhead and performance using an experimental prototype running on the ORBIT radio grid testbed. The prototype system implements a heterogeneous network with 18 autonomous systems each with a compute cluster that participates in the control plane protocol and executes specified resource sharing algorithms. Experimental results are given comparing the performance of the baseline with no cooperation to that of cooperative algorithms for compute offloading, cluster computing and service chaining. An application level evaluation of latency vs. offered load is also carried out for an example time-critical application (image analysis for traffic lane detection). The results show significant performance gains (as much as 45% for the cluster computing example) vs. the no cooperation baseline in each case at the cost of relatively modest complexity and overhead.

5.1 Introduction

This Chapter presents a novel control plane protocol designed to enable cooperative resource sharing in heterogeneous mobile edge cloud (MEC) scenarios. MEC [157, 158] is motivated by significantly lower network delay between a mobile device and a compute server, and is thus considered as a solution for low latency applications such as augmented reality, industrial control and autonomous driving [159, 160, 161, 162]. While MEC offers the advantage of potentially low latency for time critical applications, computing load generated by mobile users at the network edge can be very bursty as compared with aggregated traffic served by a data center. This burstiness in traffic demand can be overcome by pooling computing resources across multiple edge clouds in a region, motivating the design of a shared control plane to enable such cooperation. The focus of this work is thus on the design of a lightweight control protocol which provides neighboring edge clouds with visibility of their computing and network resources along with current load metrics. The design is intended to promote regional awareness of available resources in a heterogeneous multi-tenant environment so as to enable cooperative techniques such as cluster computing, compute offloading or service chaining [158, 163, 164, 165].

Figure 5.1 is an illustration of how a mobile client in an MEC service scenario can identify resources for offloading. In this example, a mobile client wishing to offload computing tasks will in general require information about the current status of available MEC nodes where the task can be executed. The most prevalent architectural solution is the central controller [166, 167] shown in the figure, which is a reasonable design for software-defined enterprise networks under the management of a single operator. However, if edge cloud services are adopted at scale, it may be expected that coverage of an entire city or region will be heterogeneous and multi-tenant in nature, involving multiple operators/owners who cannot possibly belong to a single administrative domain. Experience with the Internet shows that building a large-scale network with organic growth involves the adoption of cooperative protocols (such as the border gateway protocol, BGP [25]) between autonomous systems in a way that enables flexible business

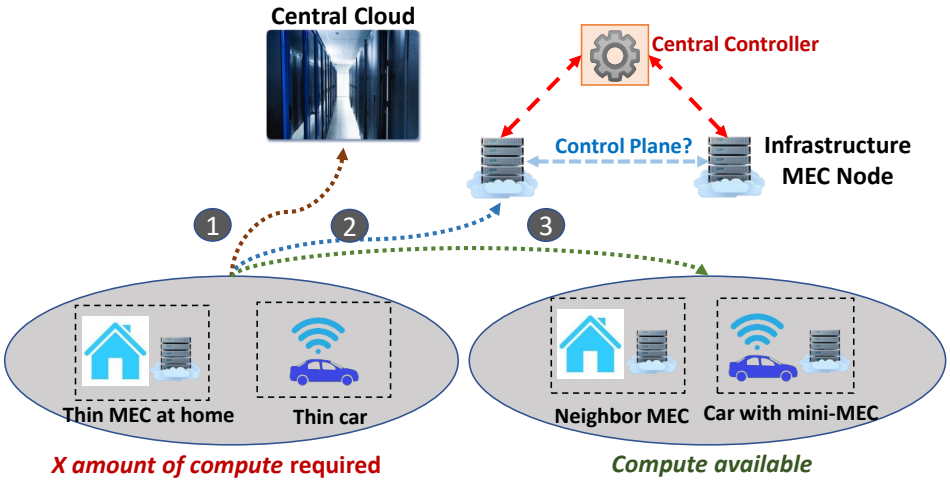

Figure 5.1: Computation Offloading to: (1) Central Cloud, (2) Infrastructure MEC node, and (3) Neighbors.

models for monetization of contributed resources. We believe that the same general philosophy applies to edge clouds, i.e. organic growth and large scale will be best promoted by a distributed control architecture which provides a lightweight mechanism for cooperation and monetization of edge cloud resources. The distributed approach avoids centralized points of control which can severely limit business model flexibility and prevent the emergence of grass roots edge cloud service providers [168, 169]*.

Figure 5.2 is a conceptual diagram of the distributed edge cloud control plane architecture proposed in this work. As shown in the figure, each set of edge cloud resources is represented by a domain controller (ECDC). Each ECDC voluntarily identifies neighbors with which it wishes to collaborate, either as a peer or as service provider. The ECDC participates in a distributed resource update protocol in which neighbors exchange status parameters (such as compute capability, load, bandwidth, etc.) in order to develop a regional map of available resources to be used to enable cooperative resource

*It is noted here that while the focus of this work is on the resource control protocol and algorithms, additional service level agreement (SLA) protocols [170, 171, 172] which are beyond the scope of this work will also be needed to support various business agreements between cloud computing peers in this architecture.

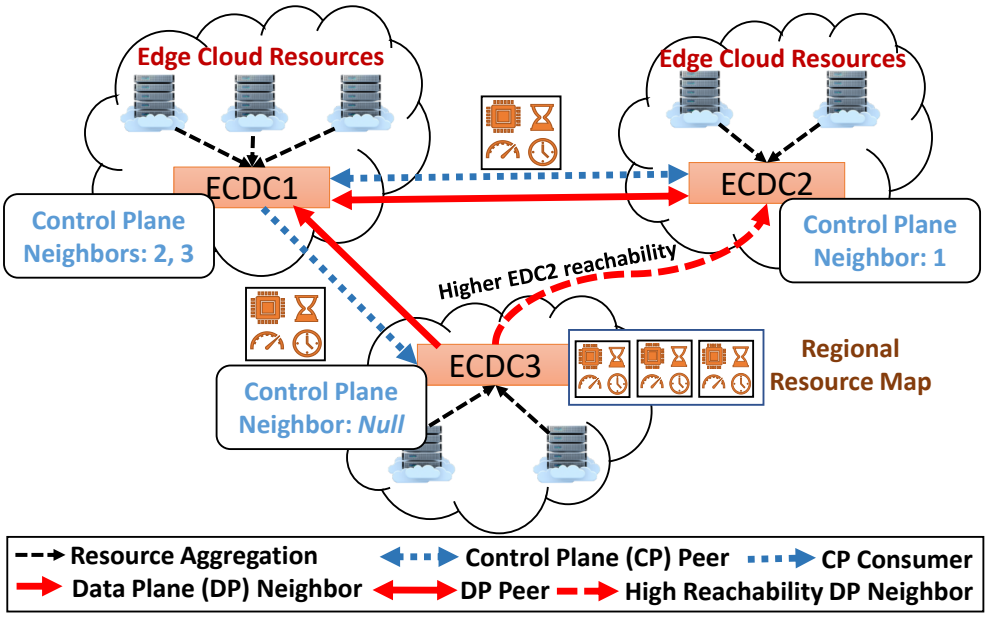

Figure 5.2: Distributed Edge Cloud Control Plane Architecture.

sharing algorithms. The scope of control information distribution can be controlled by limiting the forwarding of control packets to a specified maximum number of hops, thus limiting overheads associated with uncontrolled flooding.

The rest of the Chapter is organized as follows. Section 5.2 describes the distributed edge cloud control protocol design principles. Section 5.3 provides the implementation specific details such as configuration, information exchange phases and software functions. The cooperative resource sharing algorithms for different use-cases are discussed in the Section 5.4. The methodology to carry out large scale emulation of the proposed protocol is presented in the Section 5.5 with the details of testbed set-up and the application. Results including comparison with the centralized control plane, application independent protocol performance evaluation and analysis of a low-latency application using the proposed protocol, are presented in the Section 5.6. Section 5.7 provides a discussion on the related work in this field. Finally, Section 6 concludes this Chapter.

5.2 Protocol Design

This section introduces the design challenges and provides the protocol design details. We start with the high level design goals next.

5.2.1 Design Goals

Design for scalability and growth. Currently evolving mobile network architectures such as 5G aim to populate vast geographical areas with high-speed wireless access networks along with supporting edge cloud infrastructure. The goal is to enable grass roots models which encourage organic growth of large scale edge cloud systems.

Lightweight and compatible protocol. The protocol should be lightweight and designed as an overlay so as to avoid requiring changes to existing IP protocols or existing cloud stacks. Also, protocol overhead should be kept reasonable, avoiding problems associated with unconstrained flooding.

Distributed architecture which promotes federation and cooperation. The control plane protocol should work in a fully distributed manner with edge cloud nodes/clusters as peers. The protocol should work across multiple ownership domains and encourage sharing/monetization of regional computing resources.

Flexibility for local policy choice. Each edge cloud node should be able to choose its own number of neighbors and can decide independently how far in the network its own information should propagate.

Heterogeneity support. The protocol should capture key parameters associated with heterogeneous computing clusters with varying network quality.

Timely dissemination of control information. Resource status updates should be timely enough to enable sharing algorithms at the granularity of a single task request.

5.2.2 Protocol Design

In this section, we present the protocol design used in DISCO, a distributed control plane protocol capable of handling heterogeneous compute nodes for their offloading requirements.

i. Control Plane Information

The periodic control plane messages are used to gather the capabilities of neighboring edge cloud node which can be later used to offload the tasks. From an edge-centric perspective, a node must be aware of: (a) the network distance (e.g., Round Trip Time – RTT), (b) network capacity (e.g., bandwidth), (c) node computation capability (e.g., Giga floating point operations, per second – GFLOPS), and (d) current utilization of the node, of its peer. Along with these, the EC specifications (e.g., number of processors, number of cores, processor architecture, machine type etc.) may be a part of the control plane information. While the EC specifications, (c) and (d) can be obtained directly from the node's control plane information packet, (a) and (b) are measured by establishing a direct connection with the peer.

Design Implications. As the information about all the peers is locally maintained at each node, the node can independently decide which peer to choose for computation offloading. Moreover, this essential information set can assist a node to make a *best* decision for use-cases such as cluster computing, compute offloading, or service chaining.

ii. Information Exchange

The control plane information is distributed from each edge cloud node to its *agreed* peers. The agreement can be peer-to-peer in which both edge clouds agree to send the control messages to each other or peer-to-consumer where one edge cloud is the sender while other is the receiver. Moreover, the information dissemination can be intra- or inter-domain depending upon the agreements. Figure 5.3 shows an example edge cloud information dissemination for the Autonomous Systems (ASes[†]) in the San Francisco (SFO) city obtained from CAIDA [1]. It is noted that a consumer for one AS can become peer in another agreement and vice-versa. Each edge cloud has a pre-defined number of directly agreed peers (N).

Design Implications. The agreement based control plane information exchange design avoids network flooding. Also, the node can independently decide to become a peer or

[†]Note that edge cloud domain controllers need not have a 1:1 mapping with ASes as in this example. Each AS will have at least one ECDC, and typically many more for a large AS spread across a wide geographic area.

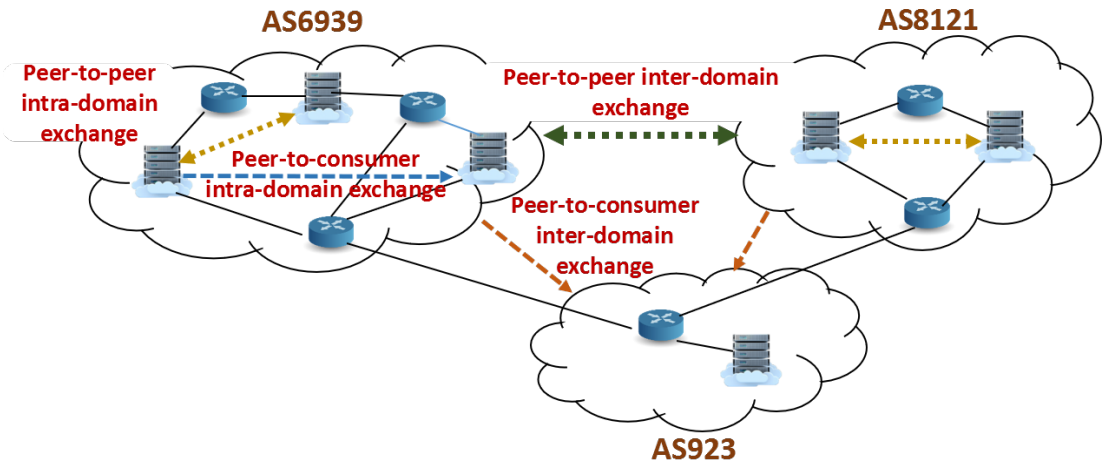

Figure 5.3: Information Dissemination in *DISCO*

a consumer depending upon its own computation and other resource availability.

iii. Information Propagation

The information is recursively propagated from an edge cloud node to the peer-of-peer or consumer-of-peer. This progressive control plane messaging enables an edge cloud to send its capability information to unknown but trustworthy edge cloud nodes who can then offload computation to it. In $DISCO$ control packet, maximum number of hops ($MHops$) corresponding to the control plane propagation depth in the network and, current hop count ($CHops$), are embedded. $CHops$ is set equal to the $MHops$ at the control packet generation node which is decremented at each edge-hop, discarding the packet when it becomes zero.

Design Implications. The edge-to-edge hop design allows the information propagation through trusted neighbors. The $MHops$ provides the generator node complete control to regulate its information reachability. The design corresponds to the telescopic flooding [173] where if $MHops$ is set too high (greedy), the control information will be delayed at a far edge cloud node. Moreover, such $MHops$ settings will discourage the farther edge cloud node to offload their computations for which a service-level agreement (SLA) might not exist. The $MHops$ of a node also helps its own reachability for cooperation.

iv. Periodic Control

The periodicity of control message generation can be independently set by the edge cloud node. The period can be determined by a node based upon its timescale of resource variability.

Design Implications. The independent nature of periodicity avoids the need of global synchronization. Moreover, as the node capabilities vary, the periodicity can be changed accordingly, thereby updating the direct neighbors immediately.

5.3 Implementation

This section describes *DISCO* protocol implementation using Linkpack [174], Click modular router [175], OpenCV[48], Java and Python.

a. EC Node Resource Measurement

Compute Capability. Compute capability of the edge cloud node is benchmarked in GFLOPS using Linkpack version 3.011. Linpack loads the node with a dense n by n system of linear equations $Ax = b$ to measure its computing power.

EC Utilization. The Linux *mpstat* is employed to periodically measure the current utilization (between 0-1).

EC Specifications. The EC node machine specifications are parsed from the $/proc/cpuinfo$, obtaining machine type, number of cores, threads per core and processor frequency.

b. Peer Configuration

Each ECDC maintains a configuration of its first hop *agreed* peers as shown in red color in Figure 5.4, preserved as an overlay over the physical network configuration. The peer configuration is independently set by each ECDC. In case a node does not have sufficient compute capabilities or does not want to receive data plane offloads, it can set peers as *null*. The nodes listed in peering configuration are the first set of neighbors to receive the control plane packet from a node.

c. Control Packet Generation

The *DISCO* control packet designed as an overlay on top of IP consists of the

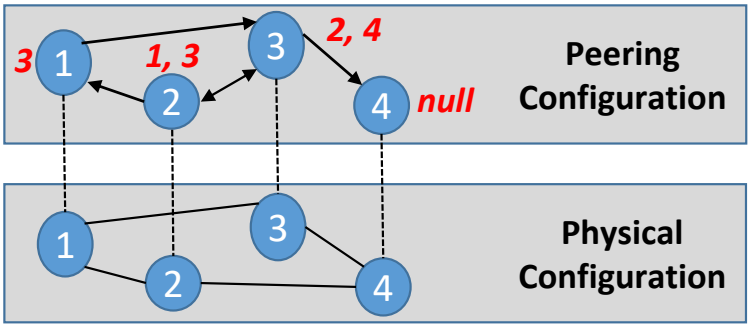

Figure 5.4: Peering Configuration for Control Exchange.

following fields as shown in Figure 5.5. The *Type* (1 octet) denotes that its an EC control packet. The $CHops$ (1 octet) and $MHops$ (1 octet) denote the current and maximum number of hops. Initially $CHops$ is set equal to $MHops$ and is decremented by one at each of the ECDC until $CHops = 0$, when the packet is discarded. The $SECID$ (20 octets) and $DECID$ (20 octets) are the source and destination globally unique EC identifications for the ECDCs. The uniqueness of these IDs can be achieved using a name certification server as described in [16, 176]. The field U (4 octets) denotes the current utilization of an edge cloud, periodically retrieved through the Linux *mpstat* command and CC (4 octets) is the computational capability of an edge cloud measured in GFlops obtained using *Linpack* during the bootstrap phase. The EC specifications field is a variable sized field used to communicate the machine specific parameters.

Why both $CHops$ and $MHops$. As the topology is not exposed to the neighbors (security) and the scheme does not rely upon the acknowledgements (timeliness), these two hop counts are used to determine and adjust the self-reachability.

d. Control Plane Information Exchange

$DISCO$ is designed with the following phases for control plane information exchange.

i. Bootstrap Phase Bootstrap phase is an initial phase which occurs when the ECDC/cluster is born in the system or undergoes any changes during its lifecycle. In general, the following events are associated with the bootstrap phase:

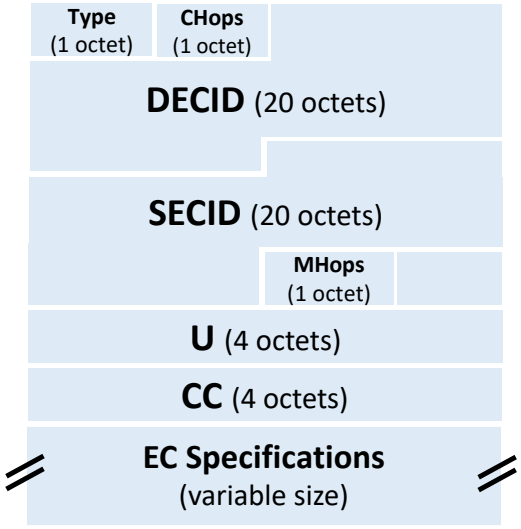

Figure 5.5: Control Packet Format in *DISCO*.

1. *Determine Specifications*: In this step, the machine specifications such as number of processors, number of cores, processor architecture, and machine type are obtained.

2. *Estimate Compute Capability (CC):* This step estimates the number of GFlops of an EC node without any load. The time taken to measure CC varies for each EC node.

ii. Sending Phase

During the sending phase, a control packet is created as per the format shown in Figure 5.5. The forwarding function is also a part of sending phase wherein the control packet received from peers are forwarded to a node's own neighbors. The EC node utilization, U, is measured in this phase, while the other fields are filled using the stored values (see Figure 5.6, left). The current utilization is periodically estimated in time $t1$. The timer value for U is a function of node's resource variability and therefore can be independently set by each EC node. The control packet is created periodically in time $t2$ using the stored CC, and EC specification values. In case of forwarding a peer's packet, only the $DECID$ and $CHops$ fields are changed. The $DECID$ field is

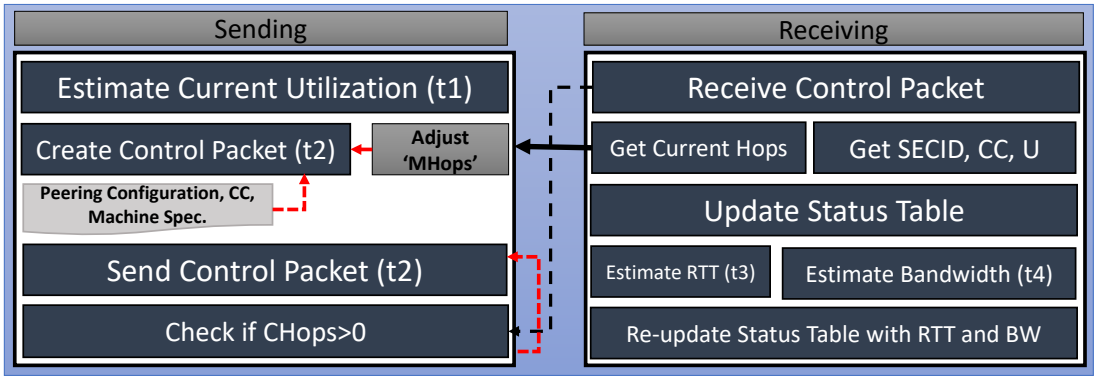

Figure 5.6: Sending and Receiving Phases in *DISCO*.

filled from the node's own neighbors while the $CHops$ is decremented by one. Control packet creation and sending occur periodically in time $t2$. The sending phase is also dependent on the receiving phase for fine-tuning the $MHops$ value.

iii. Receiving Phase

During the receiving phase (see Figure 5.6, right), if $CHops$ is greater than zero, the sending phase is triggered to forward the packet to its neighbors. The values of other parameters such as $SECID$, CC and U are also obtained and are updated in the status table as shown in Figure 5.7. The $CHops$ and $MHops$ values are used in this phase to determine the number of hops an indirect neighbor is away. For example, consider that an EC node receives a packet from a neighbor with $MHops = 4$ and the current decremented hop count, $CHops = 0$. If the maximum hop count set by this EC node, $MHops = 2$, then it is likely that the neighbor in discussion will not receive its control plane information (if the node is not in the neighbor list of itself and first hop neighbors). This simple intuition is used to adjust local $MHops$ value to increase the reachability if sufficient resources are available.

Network Resource Measurement. An important step during the receiving phase is to measure the round trip time and bandwidth to all the direct and indirect neighbors. Since the neighbors are identified using their globally unique identifiers, a name resolution server similar to [16, 176] is used to obtain the network address of EC nodes

ECID	CC	U	RTT	Bandwidth
E31	3	0.4	30ms	2.1 Mbps
-	-	-	-	-

Figure 5.7: Status Table in *DISCO*.

from their ECIDs. The round trip time is estimated using *Ping* command running periodically in time $t3$. Similarly, the bandwidth is measured using *iperf* in every $t4$ time period. As bandwidth measurement is an expensive function, in general, $t4 >> t3$.

Update EC Status Table Each EC node maintains a status table which lists all possible edge computing peers with their compute capability, current utilization, ping based network distance and the bandwidth. The information stored in the status table is used in various resource sharing strategies during the data plane.

e. System Details

The prototype is implemented using the Click modular router software running on a general purpose edge computing node. The element graph in Click, as shown in Figure 5.8, provides an easy to configure, modular schema which can be enabled for both sending and receiving modes simultaneously. The $FromDevice$ and $ToDevice$ elements are configured with their respective Ethernet ports. The peering configuration is set using a topology file passed as a parameter. The timer values can be set and sending mode can be periodically triggered as per the timer value $t2$. A port of the name resolution server (NRS) is used to obtain and cache the ECID to network address mapping during the first receipt of a packet for an $SECID$.

Software logic for $DISCO$ consists of a set of generation, forwarding and table elements. The packet classifier helps in identifying control and data packets which are forwarded to the respective click elements, namely, Control and Data packet logic. The EC manager class inherits generation and forwarding elements. The generation phase involves accessing the database for CC, U, $CHops$, $MHops$ and peer configuration which are updated at the respective fields in the control packets. An EC node status table is updated based on the current status information received from the control

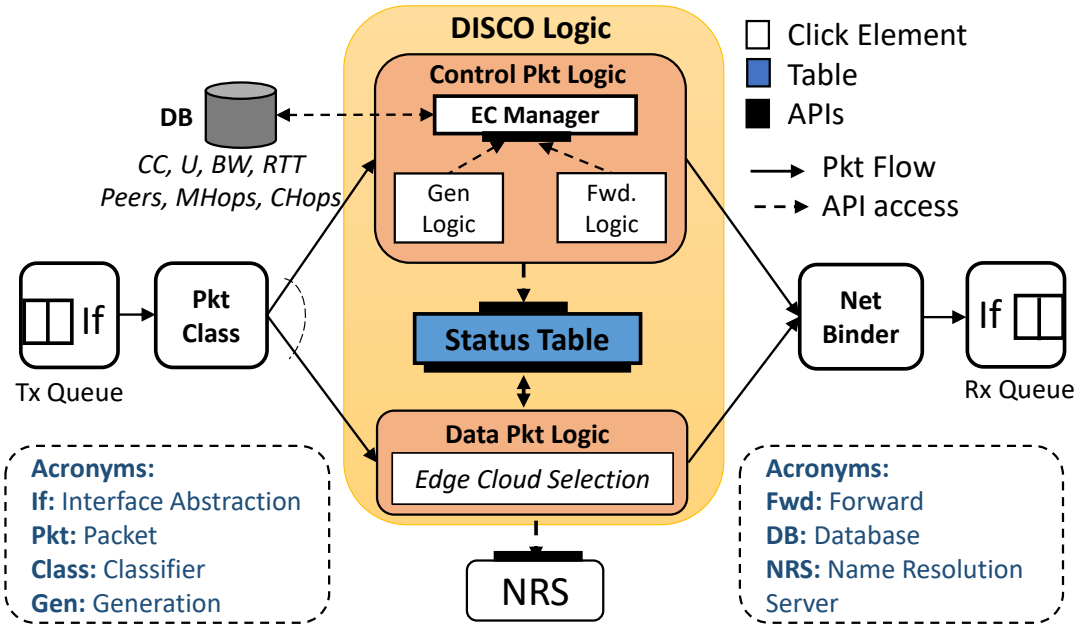

Figure 5.8: *DISCO* Node Logic Implementation

packets. The network statistics such as BW (Bandwidth) and RTT are measured using standard Linux tools for each edge peer and are updated in the status table accordingly. The forwarding phase relays the received control packets from peers based on the neighbor list while updating destination IDs and decrementing $CHops$. The packet is discarded when $CHops$ become zero.

i. Loop Removal. Packets are not forwarded on the received input port. Furthermore, if the $SECID$ matches $DECID$ from the peer configuration, the control packet is discarded without forwarding to prevent loops.

ii. Data Freshness. The received packets are first compared with the status table and if the timestamp of currently received packets is higher than that of the previously received packet for that particular $ECID$ obtained from the status table, the packet is not forwarded and discarded.

5.4 Cooperative Resource Sharing Algorithms

The cooperative resource sharing technique is used to fulfill the requirements of an application or a use-case as follows. In case a node does not have enough compute resources, it can use the computation offloading to utilize resources of another node. The faster processing of big data can be achieved by sharing a group of resources in a parallel or distributed (cluster) manner. In order to utilize a specific service available at a node to accomplish a broader task, pipeline based service processing (service chaining) allows sharing resources sequentially.

Offloading. The control plane capability extends the *DISCO* functionality to select a best EC node to offload a task and forward a data packet to that node. Different schemes for various use-cases are implemented for the performance evaluation.

For a given compute cluster, the overall compute capability of can be calculated as $G = \sum_{i=1}^{N} g_i$, where g_i is a node's compute capability in GFlops. Thus, the normalized compute capability of a node with respect to the whole system is $g_i^{norm} = g_i/G$. Assuming the current load at each node is $load_i$, we define the compute power of a node as $\gamma_i = g_i^{norm} * (1 - load_i)$. When selecting k nodes for resource sharing, the average available compute power for an EC node can be given by Equation 5.1.

$$\Gamma_i = \sum_{j=1}^{k} \gamma_j / k \tag{5.1}$$

Therefore, the objective of an MEC control plane architecture is to select nodes leading to $\Gamma = \sum_{i=1}^{N} \Gamma_i$ maximization.

Several cooperative resource sharing algorithms are available in the literature assuming partial or complete information availability at each node. Based upon the information, algorithms such as round-robin, random selection, select-all, centralized, and game-theory etc., are used [177, 178, 179].

Following an application independent approach, and depending upon the information availability, the protocol can be generally evaluated for overall system performance using one of the following approaches.

Random Selection (R). Here, among r available neighbors, $0 \leq k \leq r$ nodes are selected arbitrarily for resource sharing.

Round-Robin (RR). In this case, the nodes are chosen in a round-robin fashion. For $k > 1$, randomized group of neighbors can be selected in a round-robin way.

Optimal (O). In this case, k *best* performing nodes are selected for resource sharing. In case of same RTT and BW, the *best* nodes can be selected based upon their sorted γ_i values.

5.4.1 Application Performance Evaluation

The proposed $DISCO$ protocol can also be used to evaluate the application performance for various cases such as cluster computing, single node task offloading, and service chaining.

Cluster Computing

The batch processing tasks can be divided among the available data plane neighbors of an EC node using following schemes.

Equal-share. In this schemes, the T tasks are equally divided among all the k available neighbors as $task_i = T/k$. This scheme is bottle-necked by the straggler EC node among the k neighbors which has maximum total response time for a batch (sum of transmission, propagation & processing delay).

Proportional. In this $DISCO$ enabled scheme, the tasks are distributed proportioned among the available neighbors based upon the inverse of their normalized estimated service time. In general, if the service time of each node is s_i, $i \in (1, k)$, the task allocated to a node i from the batch size B can be calculated using Equation 5.2.

$$task_i = \Big(\prod_{j=1(j\neq i)}^{k} s_j / \Big(\sum_{i=1}^{k} \prod_{j=1(j\neq i)}^{k} s_j \Big) \Big) * B \tag{5.2}$$

For both the schemes, the maximum available neighbors based upon the peering agreements are chosen for parallelism.

Single Node Task Offloading

In this case, a single EC node is selected to process a task. The output of the task is returned to the home MEC node once processed. If the selected node is local, then there is no offloading involved. Apart from a neighbor's processing delay, the total transmission as well as propagation delays are also considered to make a decision to offload. The single node task offloading approach can be evaluated using following schemes.

Baseline. In this scheme, the processing requests are handled by local EC nodes. This scheme does not use $DISCO$ and there is no task offloading involved at any of these EC nodes.

Thresholding without $DISCO$. In this scheme, the image processing requests can be handled locally or offloaded to a neighbor if the EC response time is beyond the application latency threshold. As the neighboring node's information is not available through $DISCO$, the offloading node is chosen from the list of neighbors based upon the available bandwidth which behaves as a random selection when the inter-edge bandwidth between all the EC nodes is same.

Thresholding with $DISCO$. In this $DISCO$ enabled scheme, the processing requests can be handled locally or offloaded based upon comparing the local processing delay, and the transmission, propagation and processing delay costs of offloading to a neighbor. The offloading decision is initiated using an application latency threshold.

Best Choice (BC). In this $DISCO$ enabled best choice offloading decision scheme, the processing request is always offloaded to the *best* EC neighbor, if available.

In the context of MEC, assuming that the inter-edge bandwidth and round-trip delay are same between all the nodes, the goal is to select node(s) with minimal load and highest compute capabilities (high γ).

NFV and Service Chaining

For a NFV service chaining, the following algorithms can be used to select k pipeline nodes for subsequent task processing.

Random. The random scheme selects an arbitrary node from the list of available neighbors to offload the partially-processed task and the next neighbor after accomplishing the next task repeats the process until the final stage is completed.

Next-best Sequential. This scheme uses the information provided by *DISCO* to select a node with lowest response delay and allows the neighbor to do the same for the remaining tasks in the chain.

Global Optimal. The optimal assumes that the information about all the nodes is available at the home node beforehand and therefore selects k best nodes for service chaining. This approach resembles the traveling salesman problem [180] and has non deterministic polynomial time complexity.

5.5 Large Scale Emulation Methodology

The large scale evaluation of the proposed distributed control plane protocol is carried out using the publicly available Autonomous Systems (ASes) dataset of San Francisco (SFO) city from CAIDA [1] and inferring details from their published capacity [181, 182].

5.5.1 Setting Parameters

AS to Location Mapping

Each AS can have one or more ECs in its domain. We chose a total of 18 top ASes from the dataset and analyzed their network as well traffic capacity [181, 182]. An Internet Service Provider (ISP) may operate in multiple cities and/or countries and therefore there is no AS to geo-location explicit binding available. Thus, in this work, we assume geographical location of an AS as the origin AS's office location in the city with the assumption that the traffic for that AS should pass through the deployed ECs. For instance, AS6939 is located at the zip code 94102 with the AS name HURRICANE.

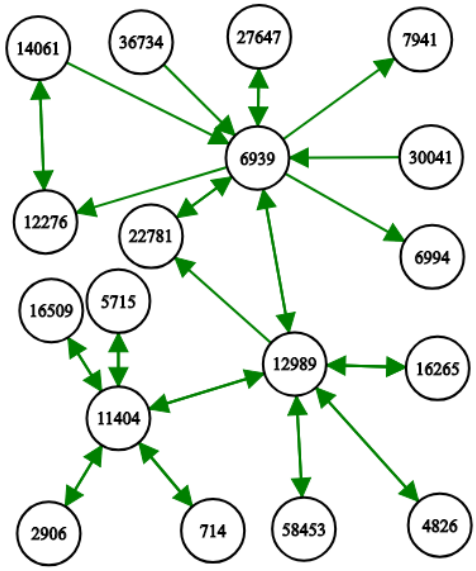

Figure 5.9: AS Relationship in the SFO (CAIDA [1])

Neighbor List

We infer the neighbor's of an AS located in SFO using the following relations: (a) peer-to-peer and (b) peer-to-consumer. As the relation is not transitive, a consumer might have other ASes peer or consumer and so on. The AS relationship of top 18 ASes in SFO is shown in Figure 5.9. The relation can be explained as follows: AS6939 has peer-to-peer relation with AS27647, it is a consumer of AS14601, and it has a peer-to-consumer relationship with AS12276. A peer sends control plane information to its peers and consumers which in turn can offload their tasks when required. The neighbor list is stored at each AS location using this relationship.

Number of Edge Clouds in an AS

The estimated traffic capacity and the number of IP addresses for each AS are obtained from the publicly available information as shown in Figure 5.10. The traffic capacity is directly related to the ability of an AS to handle amount of data to and from the

AS#	AS Name	Estimated Traffic Capacity	#IP Addresses
6939	HURRICANE	10 Tbps	526336
6994	Fastmetrics	1-5Gbps	10752
7941	Sound Advice Limited	NA	6144
12276	SF Metropolitan Internet Exchange (SFMIX)	NA	256
22781	Strong Technology LLC	5-10 Gbps	35840
27647	Weebly	50-100 Gbps	2304
---	---	---	---

Figure 5.10: Example AS Information in SFO

network. Therefore, for the emulation, we assign the number of ECs available in proportion to the traffic at each AS. The number of ECs in our emulation are between from one and seven chosen to match our evaluation capabilities on the ORBIT testbed [86]. For example, AS6939 is assumed to have seven ECs.

Maximum Number of Hops

The average AS path length is proportional to the number of hops. This implies that the AS which propagate its information to a farther location should hold more IP addresses. Therefore, the maximum number hops, $MHops$ is proportioned to the number of IP addresses available at an AS. It is noted that with $MHops = 3$, all the ASes can be reached with the given AS relationship in Figure 5.9. For the evaluation, the $MHops$ is set between 0 and 3 where $MHops = 0$ implies that the AS is consumer (stub AS) and cannot handle the offloaded tasks; thus it does not propagate its own as well as others control plane information to any other node.

5.5.2 Tesbed Set-up

The prototype is evaluated at the ORBIT [86] testbed using heterogeneous compute nodes.

EC Nodes

We selected 18 grid nodes in ORBIT with varied compute capabilities and measured their raw GFlops. The GFlops are in the range of 98–194 as measured using the

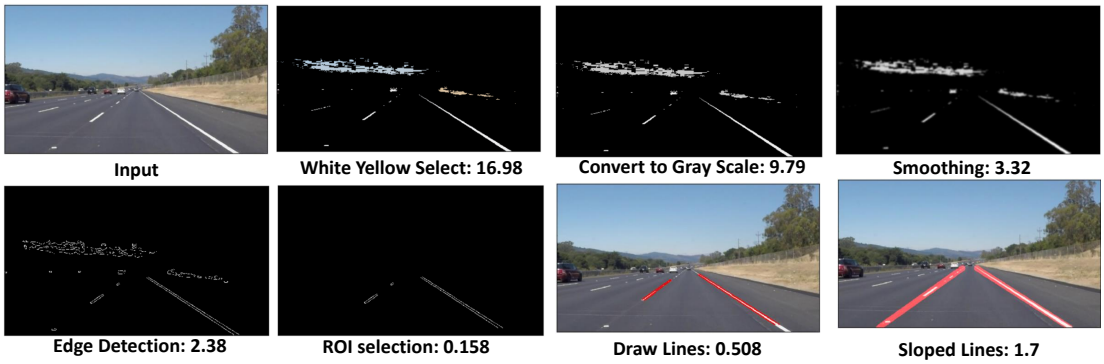

Figure 5.11: Processing Latency (ms) of Traffic Lane Detection Application

Linpack [174] tool. The nodes have different machine type, processor architectures and generations. The number of threads per core are between 1–2. The CPU processing is between 3.1–3.6 GHz.

5.5.3 Application Details

We used a low-latency traffic lane detection (TLD) application [183] for performance evaluation. The application takes an input of image, selects white and yellow, converts to gray scale, smooths, detects edges, selects region of interest (ROI) and finally draws sloped lines on top of the traffic lanes. On an Intel i7-3770, 3.4 GHz, 4 cores local compute node, the processing latency of different steps of this applications are determined as shown in Figure 5.11. The baseline performance of TLD is obtained at different EC ORBIT nodes as shown in Figure 5.12. It is observed that with higher GFlops, the processing latency of TLD can be less than 55 ms. It can be noted that for an EC end-to-end evaluation, the network latency including transmission, propagation and queuing should also be considered.

Figure 5.13 shows the variation of TLD processing latency with EC node load and it GFlops. This mapping is stored at an EC node to estimate the application performance once the CC and U are obtained through the $DISCO$ control plane protocol. Although an EC node can choose the neighbor with minimal processing latency, other parameters such as RTT, BW and peering relationship to the neighbor are also taken into account

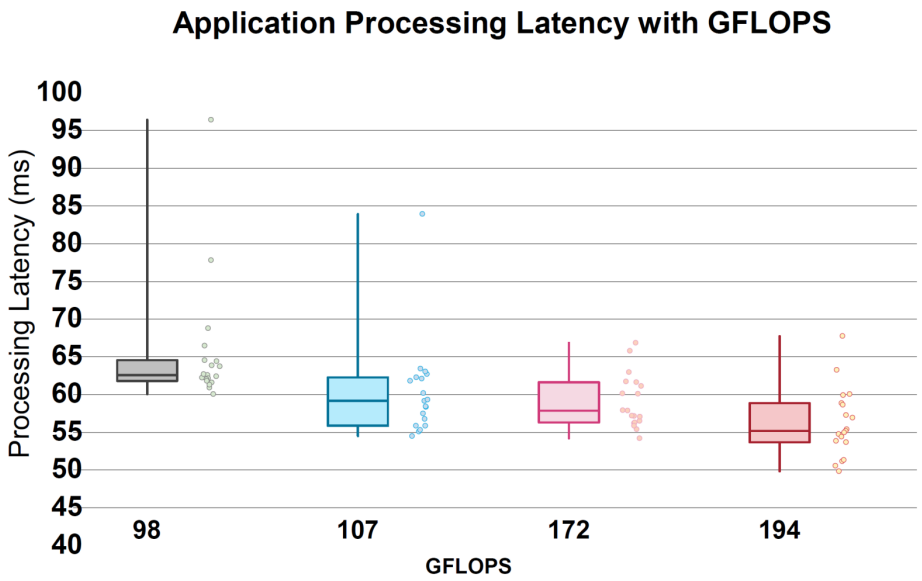

Figure 5.12: Processing Latency of Traffic Lane Detection Application with Different GFlops

for better performance.

5.6 Performance Evaluation and Results

In this section, we present the protocol evaluation results for: (a) centralized and distributed control plane overhead comparison, (b) *DISCO* protocol performance, and (c) application performance using *DISCO*.

5.6.1 Control Plane Overhead Evaluation

The control plane information can be disseminated in at least two ways. In one approach, a central controller such as SDN periodically queries the resource status of EC nodes and then multicasts or broadcasts this information. Alternatively, the information systematically propagates across the network on *need-to-know* basis. These approaches are evaluated by setting up a six node ring topology using the ORBIT nodes as shown in Figure 5.14.

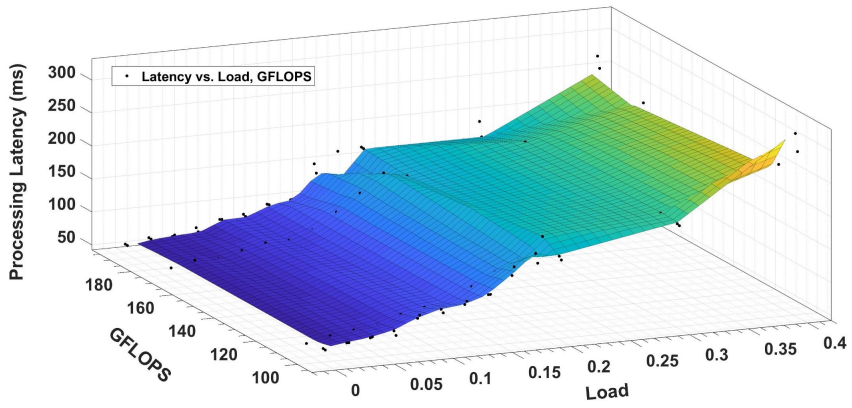

Figure 5.13: Processing Latency of Traffic Lane Detection Application with Load and GFlops

The six routers are connected in ring fashion each associated with an EC node. EC node has a unique ECID. The inter-router bandwidth is 932 Mbps. The central controller collects and disseminates the control plane packets periodically from and to each of the EC nodes, respectively. The $DISCO$ protocol runs as an overlay with each EC node setting its own peering configuration and $MHops$. For both the cases, packets are captured using the *pcap* filter. The control packet transmission interval (CPTI; timer, $t2$ in 5.6) is varied from 0.1–10 seconds. For $DISCO$, the average number of neighbors, N is varied from 0.5–5 and $MHops$ is varied from 1–5. As the loop removal logic in the implementation discards the redundant packet, $MHops$ cannot be set beyond the maximum network depth.

Packet Overhead

Figure 5.15 shows the impact of varying number of hops, CPTI and average number of neighbors on the packets generated per second in the system. Here, $N = 0.5$ implies that every alternate EC node has a neighbor or one EC has three neighbors and others have zero which are chosen randomly to satisfy the average requirement. For $N = 0.5$, when $MHops$ is increased from two to five, we observe that there is no rise in the packet overhead as the network depth is lesser than the maximum hops set by an EC

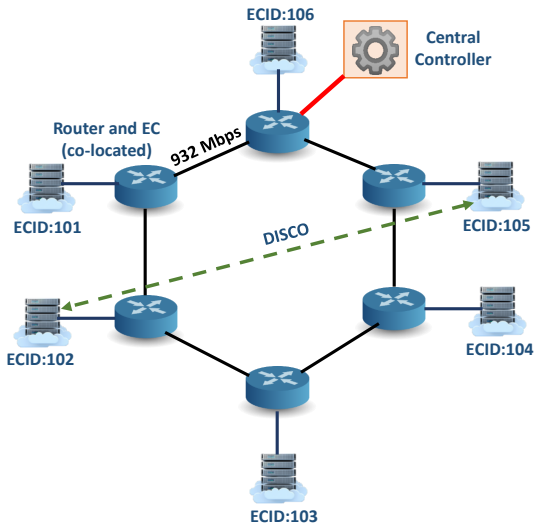

Figure 5.14: Ring Topology for Centralized vs. Distributed Control Plane Overhead Evaluation.

node. Similarly, lowering $MHops$ has significant improvement on the number of control packets generated as observed for $N = 5$ which implies that each EC node maintains all five other ECs as its peer. Therefore, in such a case, setting $MHops$ as one is necessary and sufficient to propagate control plane packet throughput the system.

The tradeoffs between N and $MHops$ can also be observed from the application point of view. For example, if an EC node has a large number of neighbors, setting $MHops$ too high will mean that the SLAs will not be met for all the offloading requests. In our protocol design, configuring neighbor lists along with setting $MHops$ and timers, are left to the provider. Finally, varying CPTI has definite impact on the packet overhead and can be systematically adjusted depending upon the node's resource variability. In general, an EC can also decide to generate the control plane packets only when the node's resource changes, and otherwise operate as a relay to forward control packets originated from its peers.

The centralized approach depends only on the CPTI value for the packet overhead and cannot be controlled by an EC node. Assuming that each EC node periodically sends control packets to the central controller and the controller forwards the packets to all the EC nodes, the packet overhead is observed to be proportional to the number of

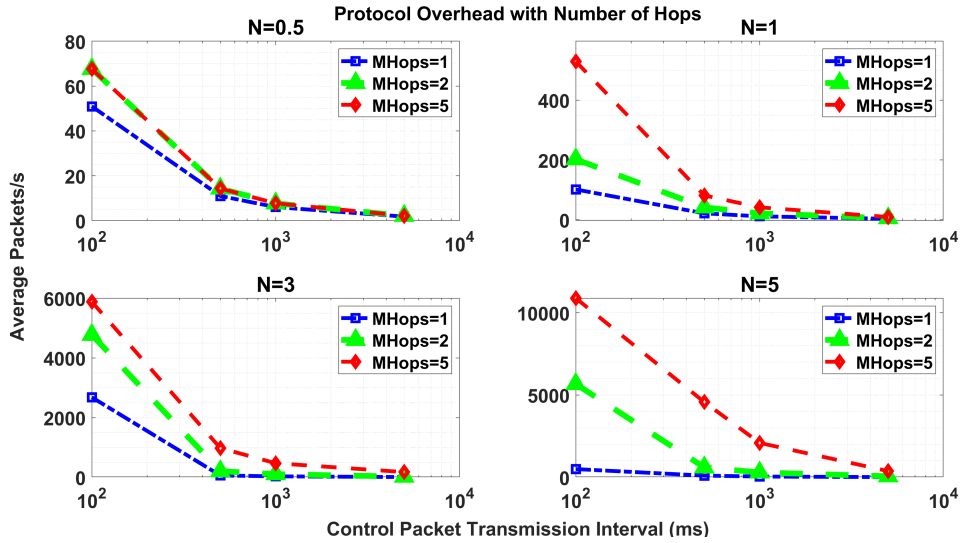

Figure 5.15: *DISCO* Packet Overhead with Varied Number of Hops and Neighbors

participating EC nodes. In particular, the total number of overhead packets per second can be calculated as $2 * M * \frac{1}{CPTI}$, where M are the total number of EC nodes in the system. It is assumed that the central controller aggregates information from all the nodes and sends as a single packet to all the nodes. For $CPTI = 1$ seconds, the total number of packets generated are therefore 12 in this system which are more than the case of $DISCO$, $N = 0.5$ and $MHops <= 5$. It can be noted that for a distributed technique, the packet overhead is a function of resource variability and therefore can be limited by a provider's parameter settings.

Convergence Time

The convergence time of a control plane protocol is defined as total wait time at an EC node to receive all the intended control packets. For a centralized scheme, the convergence time is the total time accounted for: receiving control packets at the central controller from all the nodes in the network, creating an aggregate packet at the controller, and receiving the aggregate packet at all the participating EC nodes. For $DISCO$, since $MHops$ and N are set by the individual EC nodes, accurate determination of intended control packets is infeasible. For topology shown in Figure 5.14, the

intended control packets in *DISCO* at an EC node are calculated by mapping *MHops* and N of all the EC nodes. For example, if EC101 can be reached by EC102, EC104 and EC106 in any number of hops as set by each of these EC nodes, the convergence time is the wait time to receive control packets from all these nodes.

We measure the time taken to collect all the control packets at each EC node using *DISCO* and average time (ten runs) for different *MHops* and average number of neighbors is shown in Table 5.1. It is observed that for lower N, increasing *MHops* does not increase the convergence time as there are no peers available to forward the packet. Similarly, for higher N, as all the peers are directly connected, increasing *MHops* does not increase the convergence time. For the centralized implementation, the average convergence time for the same topology is measured to be 2.78 ms which is more than the *DISCO* approach due to central controller's requirement to gather all the information before aggregation.

Table 5.1: Average Convergence Time (ms) for *DISCO*

N	**0.5**	**1**	**2**	**3**	**4**	**5**
MHops 1	0.97	0.96	1.19	1.31	1.37	1.54
MHops 2	0.94	1.49	1.43	1.93	1.54	1.54
MHops 3	0.94	1.59	1.43	1.93	1.54	1.54
MHops 4	0.94	1.74	1.43	1.93	1.54	1.54

5.6.2 *DISCO* System Level Performance Evaluation

The neighbor information gathered using the *DISCO* protocol is valuable for general as well as specific use-cases. For general case, we evaluate the application independent performance for *DISCO* using the topology shown in Figure 5.9 set-up on ORBIT radio testbed. The sum total of compute capability for this topology is $G = 2653$. Using the method described in 5.4, we calculate Γ for different maximum k values. If a node does not have k neighbors, it selects the maximum available neighbors. Table 5.2 presents the comparison between random, round-robin and optimal approach of neighbor selection for a single run. For long-run, the round-robin approach tends to

perform same as a random approach. It can be noted that only the optimal approach uses the *DISCO* protocol information. In all the cases, the information obtained using our proposed protocol ensures better performance in terms of the available compute power for the chosen nodes.

Table 5.2: Compute Power Comparison for Different Resource Sharing Algorithms, Load and MHops (k=1)

	MHops=1, Load			**MHops=2, Load**		
Algo.	**0.2**	**0.5**	**0.7**	**0.2**	**0.65**	**0.65**
R	0.91	0.57	0.39	0.9	0.53	0.39
RR	0.89	0.56	0.38	0.86	0.54	0.37
O	1.01	0.63	0.43	1.05	0.66	0.46

5.6.3 Application Performance Evaluation

The information obtained through *DISCO* can be used to enhance traditional services such as cluster (parallel as well as distributed) computing. For the AS topology shown in Fig. 5.9, the performance of the TLD application is evaluated for the same as well as variable load and inter-edge bandwidth cases. In all the cases, the load is introduced using the *stress* function in Linux and the bandwidth is varied using Linux *tc* command. It is assumed that the execution environment is already available at all the ECs.

EC Response Time. The EC response time (t_{res}) is given by the Eq. 5.3. Here, t_{proc} is the task processing time at an EC, and $t_{tx,task}$, $t_{prop,task}$, $t_{prop,task}$ and $t_{prop,res}$ are the transmission and propagation delays, for the task and its processed response, respectively. For a task executed locally (no offload) t_{res} is equal to t_{proc}.

$$t_{res} = t_{proc} + t_{tx,task} + t_{prop,task} + t_{tx,res} + t_{prop,res} \tag{5.3}$$

Cluster Computing

In order to evaluate parallel cluster computing performance, we create a Hadoop-like set up for TLD application with 100K batch jobs. The tasks are divided for the schemes presented in Section 5.4.1. Figure 5.16 compares the cluster computing scenario with

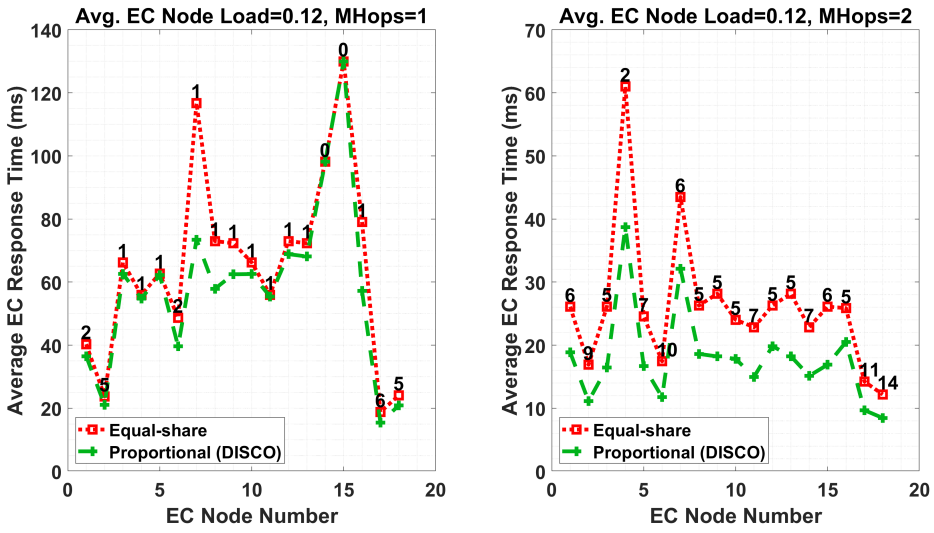

Figure 5.16: Cluster Computing with and without DISCO for Load=0.12 and Different MHops; Numbers in black are data plane neighbors for each EC node.

heterogeneous load, inter-edge bandwidth and resource distribution for EC response time. For the average load of 0.12 and MHops of one, when there are no neighbors available, the performance of equal-share and proportional task distribution is same. While having more neighbors is beneficial for both the schemes, the equal-share is always bottle-necked by the straggler whose compute capability, load, inter-edge bandwidth and round trip propagation delay information was not available during offload decision. As the *DISCO* makes a node aware of the estimated processing delay of the neighbor, sending the number of tasks inversely proportional to the delay of a neighbor improves the overall delay incurred for batch processing. For the same load of 0.12 and Mhops increased to 2, an EC node can distribute the tasks to more neighbors. Thus, in this case again, the *DISCO* based proportional task distribution provides lesser average processing delay for each EC node as compared to the equal-share.

Figure 5.17 shows the aggregate performance of equal-share and proportional schemes for different loads and MHops. Although, increasing MHops from one to two improves the latency performance for both the schemes, it is observed that for medium loaded system, increasing MHops provides larger average latency gains as compared to the low and high loads cases. This is due to the fact that at the low-load, the EC node with

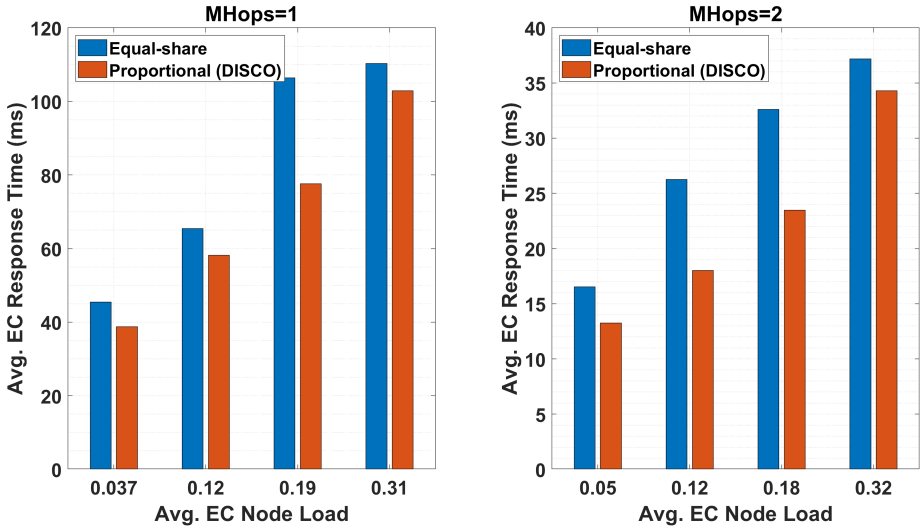

Figure 5.17: Cluster Computing Comparison for Schemes with and without *DISCO* for Heterogeneous Load, Bandwidth and Compute Resources.

equal share can also find low-loaded nodes with decent response time and at high-load, both equal-share as well proportional cannot avail low-latency as needed. However, in both the cases, proportional scheme with *DISCO* provides better performance as compared with equal-share.

Same Load and Inter-edge Bandwidth

Next, we evaluate the performance of the TLD application for offloading to a single neighbor. The evaluation is carried out from the EC's perspective, without considering the UE to EC access latency. We use a local MPEG-4 video stream of 960 pixels by 540 pixels to be processed to detect the traffic lanes. The maximum inter-edge bandwidth is fixed to 112 Mbps because of ORBIT capabilities. From the *RTT* and inter-edge bandwidth information obtained through *DISCO*, we estimate the transmission and propagation delays for all the EC neighbors. Furthermore, the processing latency is estimated using Figure 5.13 for a given neighbor's load and compute capability. Finally, the EC node is selected based upon one of the schemes explained in Section 5.4.1.

In this evaluation, the EC nodes have same load levels and same inter-edge bandwidth among nodes. The emulation is carried out for different bandwidth and load

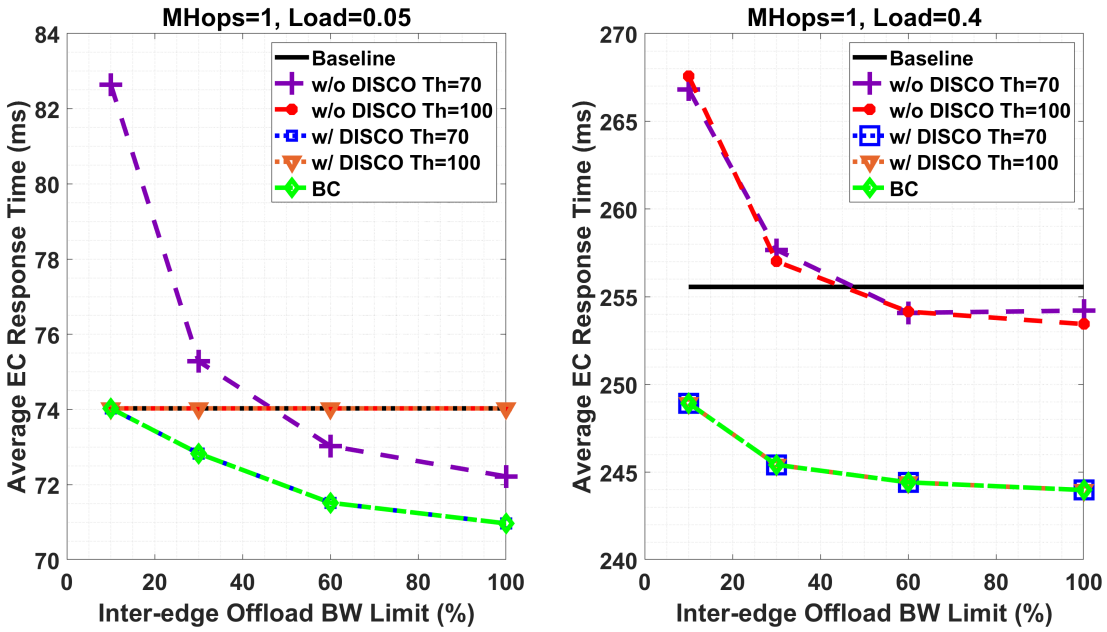

Figure 5.18: Average EC Response Time Comparison for Different Load Conditions; MHops=1

values. Figure 5.18 presents the application performance in terms of average EC response time for MHops=1 and two extreme load conditions. The inter-edge offload bandwidth limit presents the available bandwidth between pair of neighbors. For the no offload baseline case, bandwidth has no impact on the response time. In case of without *DISCO* and threshold of 70ms, the EC node attempts to offload the task to a neighbor with bandwidth as a metric which is uniform for all the pairs. The lower inter-edge bandwidth leads to higher EC response time. Therefore, for a homogeneous network, a scheme without additional control plane information works as a random selection scheme. As more bandwidth becomes available, the thresholding scheme outperforms the baseline scheme. The lower threshold limit implies that more tasks are offloaded to neighbors while the higher threshold limit at a lower load is same as baseline with no task offloading. Similarly, as the *DISCO* information is used with the threshold based task offload trigger, for lower loads and higher threshold, the scheme is same as baseline while for lower thresholds, it performs as the best choice which is an always offload (if *best* is available) scheme.

In case of the higher load (Figure 5.18, right), again, the baseline is an average scheme while the without $DISCO$ based threshold scheme improves with the bandwidth. Both low and high thresholds are triggered due to higher loads at the EC nodes but the limited bandwidth slows the offloads. Moreover, due to randomness in the selection of a neighbor, the performance is not similar. As the higher bandwidth becomes available, again the thresholding without $DISCO$ outperforms the baseline scheme. The $DISCO$ based thresholding schemes performs same as best choice because a high performance neighbor is always chosen once the threshold is triggered.

Figure 5.19 compares the application performance for different EC load and $MHops$ for the homogeneous network. It is observed that lower threshold scheme with $DISCO$ information performs similar to the best choice. Therefore, a simple thresholding scheme with an appropriate application threshold is sufficient to achieve performance better than baseline and other schemes. In this study, the threshold value zero is equivalent to the best choice, while the infinity threshold represents a no offload baseline scheme. Thus, a 100ms threshold offloads less often than 70ms but has comparable performance. As the network depth increases, the schemes without using the control plane information are unable to take advantage of other neighboring nodes while $DISCO$ based thresholding scheme can perform better even at the higher average load values.

Heterogeneous EC Network

A heterogeneous EC network is a more realistic scenario than homogeneous, and we evaluate this case using the AS topology shown in Figure 5.9. The traffic capacity and IP address information of these ASes are used to proportionate the bandwidth between the pair of nodes. The load at each node is chosen during the experiment run using uniform random distribution between the lower (α), and upper (β) limits.

Figure 5.20 shows the response time averaged over all the nodes and ten runs for this heterogeneous setting. The compute capability of each node is different as mentioned earlier. It is observed that the average EC response time gain using $DISCO$ is significantly higher than baseline and threshold based schemes for an extremely heterogeneous setting, e.g., when $\alpha = 0$ and $\beta = 0.4$. This is because $DISCO$ enables an

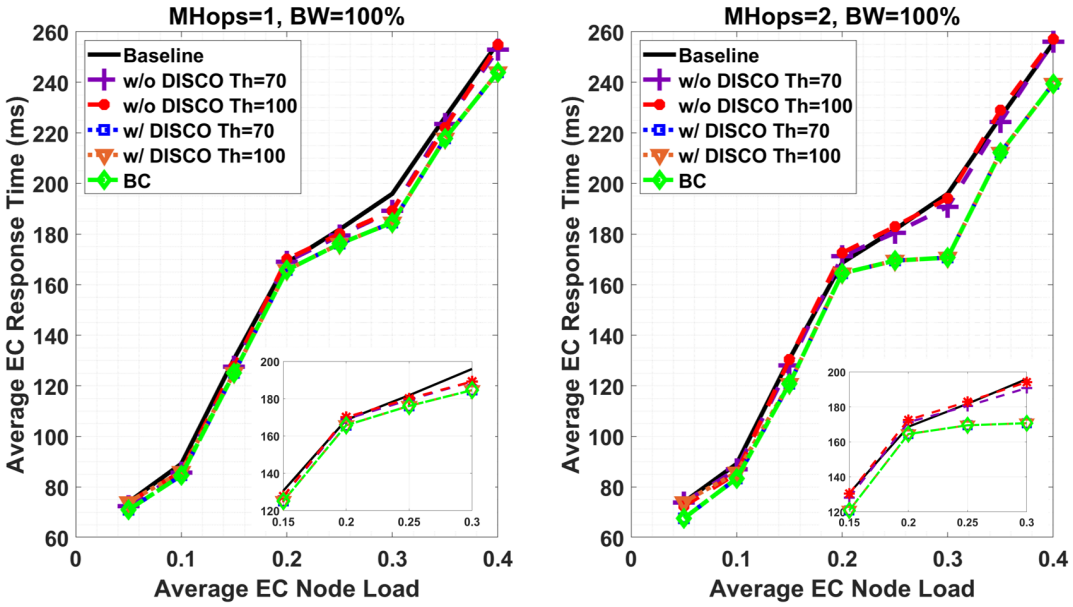

Figure 5.19: Average Response Time Comparison for Different MHops and EC Node Load

EC node to offload information to a lightly loaded neighbor which has better inter-edge bandwidth connectivity. For lower load levels, all the schemes provide similar performance as either there is no need to offload or a chosen EC node (even with the baseline scheme) has sufficient available compute capabilities.

The network depth is increased by setting $MHops$ as two and the performance results are presented in Figure 5.21. Again, using $DISCO$, the thresholding based scheme performs similar to the best choice except for very low load levels where the threshold based algorithm is not triggered. The thresholding schemes without $DISCO$ information are unable to select an optimal EC node to offload tasks. However, in some cases, particularly for the lower load scenarios, a chosen node can be good enough when just using the thresholding scheme and can outperform the baseline scheme as observed.

Figure 5.22 shows that the performance can be improved with change in the application threshold as well as MHops. For more MHops, with and without using $DISCO$, the EC node has lower response time as compared to the case with lower MHops values.

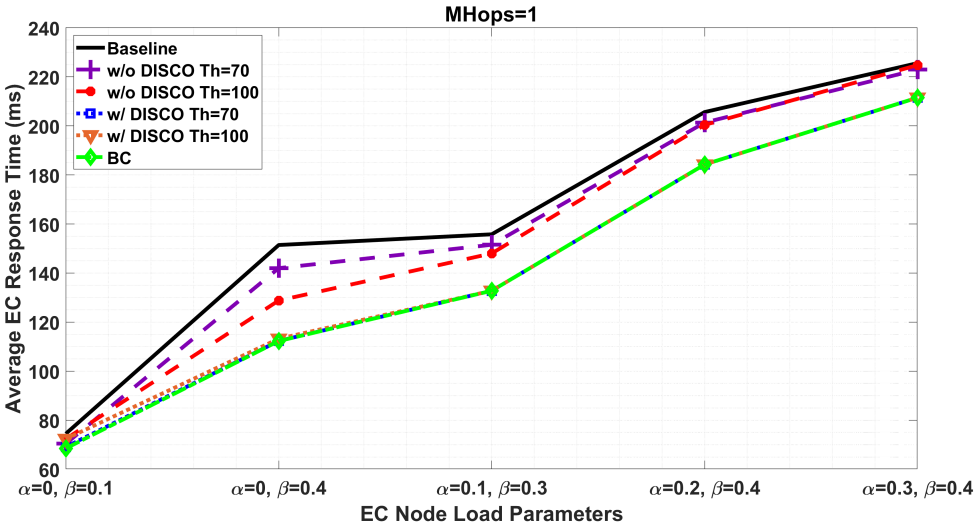

Figure 5.20: Average Response Time Comparison for Heterogeneous System Settings (Load, Bandwidth) for MHops=1

Thus, setting appropriate network depth impacts the performance. The lower latency threshold triggers more offloads which, in case of without *DISCO* schemes, does not perform well. With *DISCO*, setting the lower threshold is beneficial to trigger the offload request. Therefore when no control plane information is available, keeping application threshold higher has better performance but when using the *DISCO* protocol, setting the latency threshold lower has better performance. The low load value (0.12) used for this emulation shows that for MHops=1, not many better nodes in the given network depth are available and therefore performance improvement is not observed.

NFV and Service Chaining

The *DISCO* protocol information is also useful in an NFV service chaining use-case. We set-up a chain of TLD specific service functions (white yellow select, convert to gray, etc.; Ref: Figure 5.11) to evaluate the service chain latency performance for different schemes described in the Section 5.4.1. Figure 5.23 shows that the random scheme does not provide satisfactory results due to lack of information about a neighbor while the *DISCO*-enabled next-best scheme's performance is similar to the global optimal for all the load levels.

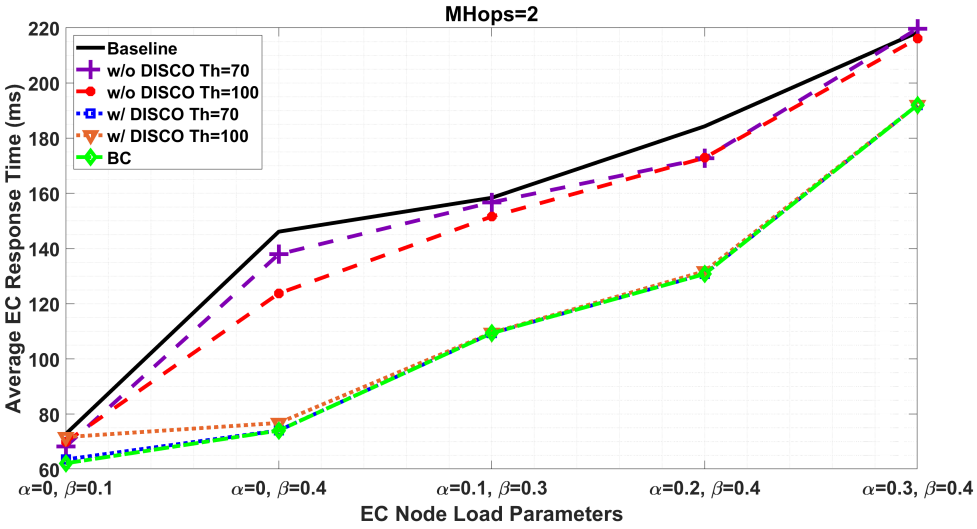

Figure 5.21: Average Response Time Comparison for Heterogeneous System Settings (Load, Bandwidth) for MHops=2

The dataset used and the project code developed for this work is made available using GitHub at the following Link [184].

5.7 Discussion and Related Work

Control Plane Variants. Many existing MEC architectures such as hierarchical [19, 20] as well as centralized [21] implicitly assume control plane information availability. To the best of our knowledge, *DISCO* is an early attempt to design, implement and evaluate the performance of a fully decentralized control plane protocol for MEC.

Overlay based Control. The overlay based control plane allows use of the existing network fabric while ensuring required flexibility, simplicity and elasticity [22]. Overlay solution are a popular choice in literature due to implementation simplicity and compatibility advantages, and is thus reflected in the *DISCO* design [23, 24].

Border Gateway Protocol. *DISCO* shares similarities with BGP which disseminates IP routes between the ASes by forming peering relationships [25]. However, *DISCO* is differentiated by its: (a) controlled network depth envisioned to scale up to a geographical region, (b) shorter periodicity as needed by MEC specifically to support

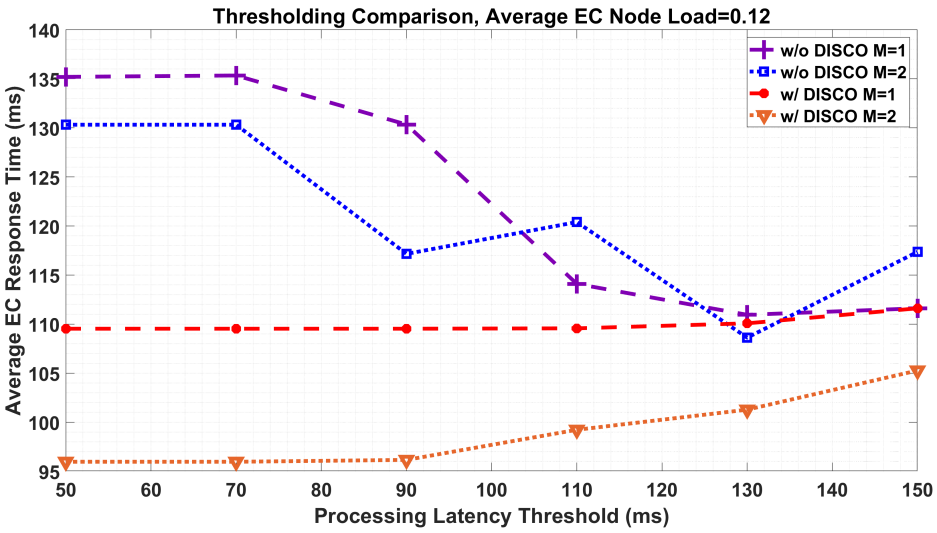

Figure 5.22: Impact of Threshold on the Application Performance for Different MHops

low-latency applications, and (c) both, network and compute information disseminate in order to enable cooperative resource management algorithms rather than simply routing.

Resource Specification (RSpec). The overlay based large-scale testbeds such as PlanetLab and GENI [26, 27] are configured for the compute and network resources using resource specifications (RSpec) [28] which enables full description of the network, control of the network topologies, and network-aware resource placement. *DISCO* is a natural complement to these testbeds for run-time resource discovery extending the current static configuration capabilities with Rspec.

Network Participation. The edge cloud network is a shared infrastructure system for MEC in which the networking entities such as routers play significant role in MEC's performance. Although the current *DISCO* protocol is built as an overlay, supporting the views of [29, 30] we believe that a tightly coupled network participation, for instance, routers disseminating network state information to the MEC nodes, will greatly improve the performance and scalability.

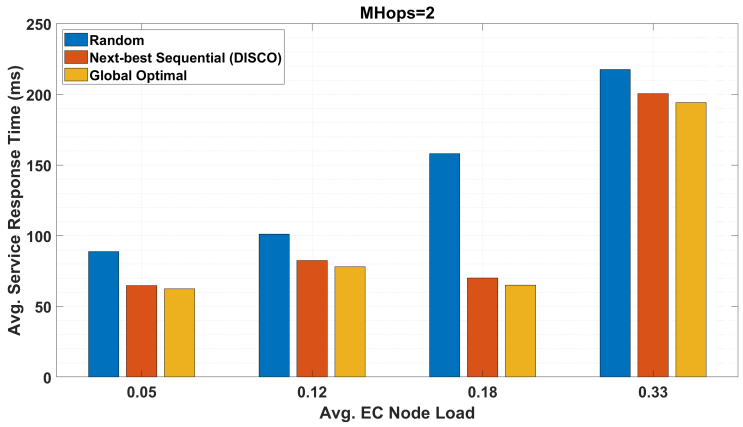

Figure 5.23: NFV Service Chain Response Time Comparison for Different Schemes & Various Average Loads.

5.8 Summary

In this work[185], a distributed control plane protocol, *DISCO*, is designed, implemented and evaluated to enable cooperative resource sharing in heterogeneous edge cloud scenarios. *DISCO* allows controlled peer-to-peer exchange of key compute and network parameters (such as CPU GIPS, % utilization and network bandwidth) needed for cooperation between heterogeneous edge clouds across network domains. The control plane information thus obtained can be used in a variety of independent data plane offloading decisions and sharing mechanisms such as cluster computing, compute offloading, load balancing multi-node orchestration, and service chaining.

The results given in this work demonstrate the feasibility of distributed control for future edge clouds, which we feel is an important new capability that will promote scalability and organic growth through grass-roots participation. System level evaluations for several use cases of the DISCO protocol further confirm that significant performance gains can be achieved through cooperation at the cost of modest complexity and overhead.

Chapter 6

Conclusions

In this thesis, we presented the architecture, design, and evaluation of the mobile edge cloud (MEC) system to support future low-latency applications. Considering the requirements posed by these applications, we proposed specific techniques in mobile edge clouds to (a) analyze the performance and scalability of low-latency applications, (b) enable application specific network routing, (c) provide service migration to handle load and user mobility, and (d) create a general-purpose control plane for the distributed resource sharing in the MEC. This thesis provided the following contributions.

First, the system-level edge cloud requirements for low-latency applications were analyzed by benchmarking a set of sample Augmented Reality (AR) applications. A city-scale MEC network simulation was carried out to evaluate the performance and scalability of the system with users running these AR applications. It was shown that the core cloud only can outperform an edge cloud only system when it has insufficient inter-edge cloud bandwidth necessary for data movement. The study concluded that it is essential to increase inter-edge cloud bandwidth along with the compute capacity to avoid network-wide congestion and support low-latency applications.

Next, using the concepts of named-object architecture, we designed a low-latency named-object based virtual network (NOVN) to realize a resource-aware anycast capability for connecting distributed MEC resources with the heterogeneous users requesting specific services. NOVN inherently supports application specific routing with minimal processing and control overhead and thereby provides Quality of Service (QoS) in a multi-tenant MEC network. The performance of NOVN is validated via experimental evaluation of an example edge cloud scenario, showing significant gains in the achievable service latency distribution.

User mobility and system load variability are key characteristics of the locally placed, geographically distributed MECs. To overcome these challenges, a service migration framework, specifically designed using containers was evaluated for a real-time application running our proposed algorithm. The application latency performance was measured at different loads for a city-scale mobility trace. This work demonstrated that container based service migration is a viable approach for handling user mobility and varying system load when parameters such as machine type, system load, inter-edge bandwidth and compute capability are considered.

Finally, a lightweight control plane protocol (DISCO) supporting the exchange of essential control information in MEC was proposed and validated with an experimental prototype. It was shown that the existence of such a control plane enables distributed resource sharing and management by providing neighboring edge clouds with visibility of their computing and network resources along with their current load metrics. Evaluating various use-cases such as cluster computing, compute offloading and service chaining, we showed that DISCO provides significant performance gains at the cost of modest complexity and overhead. We believe that the use of a control plane for edge cloud metadata will enable effective peering of edge cloud resources between multiple autonomous networks in a region.

6.1 Looking Ahead

The MEC architectural components, algorithms & protocols provided in this thesis can potentially serve as guidelines for future edge cloud deployments. Real-life use-cases such as autonomous driving, wireless AR/VR, remote surgery, and holographic teleporting are already under active consideration but many technical challenges remain. One challenge is to achieve latency of the order of a few milliseconds which is feasible with the resource assignment, cross-layer routing, container migration and control plane techniques provided in this thesis. The work presented here can be expanded across many horizons of different application profiles, network topologies, and system configurations. Further evaluation of a real-world deployment, inter-operating with different MEC implementations across multi-tenant networks remains an open challenge.

References

[1] The caida as relationships dataset. http://www.caida.org/data/as-relationships/. Accessed: 2019-01-30.

[2] VNI Cisco. Cisco visual networking index: Forecast and trends, 2017–2022. *White Paper*, 1, 2018.

[3] Guenter I Klas. Fog computing and mobile edge cloud gain momentum open fog consortium, etsi mec and cloudlets. *Google Scholar*, 2015.

[4] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, 2015.

[5] Anthony D JoSEP, RAnDy KAtz, AnDy KonWinSKi, LEE Gunho, DAViD PAttERSon, and ARiEL RABKin. A view of cloud computing. *Communications of the ACM*, 53(4), 2010.

[6] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. Cloud computing: An overview. In *IEEE International Conference on Cloud Computing*, pages 626–631. Springer, 2009.

[7] Brian D Halligan, Joey F Geiger, Andrew K Vallejos, Andrew S Greene, and Simon N Twigger. Low cost, scalable proteomics data analysis using amazon's cloud computing services and open source search algorithms. *Journal of proteome research*, 8(6):3148–3153, 2009.

[8] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. *arXiv preprint arXiv:0901.0131*, 2008.

[9] Rajiv Ranjan, Liang Zhao, Xiaomin Wu, Anna Liu, Andres Quiroz, and Manish Parashar. Peer-to-peer cloud provisioning: Service discovery and load-balancing. In *Cloud Computing*, pages 195–217. Springer, 2010.

[10] Ibrahim W Habib, Qiang Song, Zhaoming Li, and Nageswara SV Rao. Deployment of the gmpls control plane for grid applications in experimental high-performance networks. *IEEE Communications Magazine*, 44(3):65–73, 2006.

[11] Joshy Joseph, Mark Ernest, and Craig Fellenstein. Evolution of grid computing architecture and grid adoption models. *IBM Systems Journal*, 43(4):624–645, 2004.

[12] E. Bell, A. Smith, P. Langille, A. Rijhsinghani, and K. McCloghrie. Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering and Virtual LAN Extensions. RFC 2674, 1999.

[13] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031, 2001.

[14] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In vini veritas: realistic and controlled network experimentation. *ACM SIGCOMM Computer Communication Review*, 36(4):3–14, 2006.

[15] Bego Blanco, Jose Oscar Fajardo, Ioannis Giannoulakis, Emmanouil Kafetzakis, Shuping Peng, Jordi Pérez-Romero, Irena Trajkovska, Pouria S Khodashenas, Leonardo Goratti, Michele Paolino, et al. Technology pillars in the architecture of future 5g mobile networks: Nfv, mec and sdn. *Computer Standards & Interfaces*, 54:216–228, 2017.

[16] Dipankar Raychaudhuri, Kiran Nagaraja, and Arun Venkataramani. Mobilityfirst: a robust and trustworthy mobility-centric architecture for the future internet. *ACM SIGMOBILE Mobile Computing and Communications Review*, 16(3):2–13, 2012.

[17] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, 2015.

[18] Ke Zhang, Yuming Mao, Supeng Leng, Alexey Vinel, and Yan Zhang. Delay constrained offloading for mobile edge computing in cloud-enabled vehicular networks. In *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pages 288–294. IEEE, 2016.

[19] Liang Tong, Yong Li, and Wei Gao. A hierarchical edge cloud architecture for mobile computing. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

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

[21] Aditya Gudipati, Daniel Perry, Li Erran Li, and Sachin Katti. Softran: Software defined radio access network. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 25–30. ACM, 2013.

[22] An Wang, Yang Guo, Fang Hao, TV Lakshman, and Songqing Chen. Scotch: Elastically scaling up sdn control-plane using vswitch based overlay. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 403–414. ACM, 2014.

[23] Siamak Azodolmolky, Reza Nejabati, Eduard Escalona, Ramanujam Jayakumar, Nikolaos Efstathiou, and Dimitra Simeonidou. Integrated openflow–gmpls control plane: an overlay model for software defined packet over optical networks. *Optics express*, 19(26):B421–B428, 2011.

[24] Animesh Nandi, Aditya Ganjam, Peter Druschel, TS Eugene Ng, Ion Stoica, Hui Zhang, and Bobby Bhattacharjee. Saar: A shared control plane for overlay multicast. In *NSDI*, 2007.

[25] Eric C Rosen and Yakov Rekhter. Bgp/mpls vpns. 1999.

[26] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[27] Mark Berman, Jeffrey S Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5–23, 2014.

[28] Ted Faber and Rob Ricci. Resource description in geni: Rspec model. In *Presentation given at the Second GENI Engineering Conference (March 2008)*, 2008.

[29] Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. Faircloud: sharing the network in cloud computing. *ACM SIGCOMM Computer Communication Review*, 42(4):187–198, 2012.

[30] Alexander Stage and Thomas Setzer. Network-aware migration control and scheduling of differentiated virtual machine workloads. In *Proceedings of the 2009 ICSE workshop on software engineering challenges of cloud computing*, pages 9–14. IEEE Computer Society, 2009.

[31] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997.

[32] Howard Rheingold. *Virtual reality: exploring the brave new technologies*. Simon & Schuster Adult Publishing Group, 1991.

[33] Abhishek Chandra, Jon Weissman, and Benjamin Heintz. Decentralized edge clouds. *IEEE Internet Computing*, 17(5):70–73, 2013.

[34] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

[35] Jason G Caudill. The growth of m-learning and the growth of mobile computing: Parallel developments. *The International Review of Research in Open and Distributed Learning*, 8(2), 2007.

[36] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. Overlay: Practical mobile augmented reality. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 331–344. ACM, 2015.

[37] Marco C Jacobs, Mark A Livingston, et al. Managing latency in complex augmented reality systems. 1997.

[38] Wuyang Zhang, Jiachen Chen, Yanyong Zhang, and Dipankar Raychaudhuri. Towards efficient edge cloud augmentation for virtual reality mmogs. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 8. ACM, 2017.

[39] Nidhi Jain Kansal and Inderveer Chana. Cloud load balancing techniques: A step towards green computing. *IJCSI International Journal of Computer Science Issues*, 9(1):238–246, 2012.

[40] Pieter Simoens, David Griffin, Elisa Maini, T Khoa Phan, Miguel Rio, Luc Vermoesen, Frederik Vandeputte, Folker Schamel, and Dariusz Burstzynowski. Service-centric networking for distributed heterogeneous clouds. *IEEE Communications Magazine*, 55(7):208–215, 2017.

[41] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K Leung. Dynamic service migration in mobile edge-clouds. In *2015 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2015.

[42] Atheer AiR. https://atheerair.com/ .

[43] Microsoft Hololens. https://www.microsoft.com/en-us/hololens .

[44] Google Glass. https://developers.google.com/glass/ .

[45] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78:680–698, 2018.

[46] Andreas Reiter, Bernd Prünster, and Thomas Zefferer. Hybrid mobile edge computing: Unleashing the full potential of edge computing in mobile device use cases. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 935–944. IEEE Press, 2017.

[47] Dipankar Raychaudhuri, Ivan Seskar, Gil Zussman, Thanasis Korakis, Dan Kilper, Tingjun Chen, Jakub Kolodziejski, Michael Sherman, Zoran Kostic, Xiaoxiong Gu, Harish Krishnaswamy, Sumit Maheshwari, Panagiotis Skrimponis, and Craig Gutterman. Challenge: Cosmos: A city-scale programmable testbed for experimentation with advanced wireless. In *Mobicom*, 2020.

[48] Opencv. https://opencv.org/. Accessed: 2019-07-01.

[49] WiGLE: Wireless Network Mapping." WiGLE: Wireless Network Mapping. https://www.wigle.net/ .

[50] CloudPing.info. https://www.cloudping.info/ .

[51] Tiago Gama Rodrigues, Katsuya Suto, Hiroki Nishiyama, and Nei Kato. Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control. *IEEE Transactions on Computers*, 66(5):810–819, 2016.

[52] Amin Tootoonchian and Yashar Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, volume 3, 2010.

[53] 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.

[54] Albert E Fernandes Muritiba, Manuel Iori, Enrico Malaguti, and Paolo Toth. Algorithms for the bin packing problem with conflicts. *Informs Journal on computing*, 22(3):401–415, 2010.

[55] Mahadev Satyanarayanan, Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, Wenlu Hu, and Brandon Amos. Edge analytics in the internet of things. *IEEE Pervasive Computing*, 14(2):24–31, 2015.

[56] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[57] Mohammad Aazam, Imran Khan, Aymen Abdullah Alsaffar, and Eui-Nam Huh. Cloud of things: Integrating internet of things and cloud computing and the issues involved. In *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST) Islamabad, Pakistan, 14th-18th January, 2014*, pages 414–419. IEEE, 2014.

[58] Abdur Rahim Biswas and Raffaele Giaffreda. Iot and cloud convergence: Opportunities and challenges. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 375–376. IEEE, 2014.

[59] Antonio Celesti, Maria Fazio, Maurizio Giacobbe, Antonio Puliafito, and Massimo Villari. Characterizing cloud federation in iot. In *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 93–98. IEEE, 2016.

[60] Mu-Hsing Kuo. Opportunities and challenges of cloud computing to improve health care services. *Journal of medical Internet research*, 13(3):e67, 2011.

[61] Quanwen Zhu, Long Chen, Qingquan Li, Ming Li, Andreas Nüchter, and Jian Wang. 3d lidar point cloud based intersection recognition for autonomous driving. In *2012 IEEE Intelligent Vehicles Symposium*, pages 456–461. IEEE, 2012.

[62] Swarun Kumar, Shyamnath Gollakota, and Dina Katabi. A cloud-assisted design for autonomous driving. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 41–46. ACM, 2012.

[63] Luis M Vaquero and Luis Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.

[64] Jianli Pan and James McElhannon. Future edge cloud and edge computing for internet of things applications. *IEEE Internet of Things Journal*, 5(1):439–449, 2017.

[65] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jorg Ott. Consolidate iot edge computing with lightweight virtualization. *IEEE Network*, 32(1):102–111, 2018.

[66] Sandro Pinto, Tiago Gomes, Jorge Pereira, Jorge Cabral, and Adriano Tavares. Iioteed: an enhanced, trusted execution environment for industrial iot edge devices. *IEEE Internet Computing*, 21(1):40–47, 2017.

[67] Kengo Sasaki, Naoya Suzuki, Satoshi Makido, and Akihiro Nakao. Vehicle control system coordinated between cloud and mobile edge computing. In *2016 55th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pages 1122–1127. IEEE, 2016.

[68] Engin Zeydan, Ejder Bastug, Mehdi Bennis, Manhal Abdel Kader, Ilyas Alper Karatepe, Ahmet Salih Er, and Mérouane Debbah. Big data caching for networking: Moving from cloud to edge. *IEEE Communications Magazine*, 54(9):36–42, 2016.

[69] Gagangeet Singh Aujla, Neeraj Kumar, Albert Y Zomaya, and Rajiv Ranjan. Optimal decision making for big data processing at edge-cloud environment: An sdn perspective. *IEEE Transactions on Industrial Informatics*, 14(2):778–789, 2017.

[70] Ravishankar Ravindran, Xuan Liu, Asit Chakraborti, Xinwen Zhang, and Guoqiang Wang. Towards software defined icn based edge-cloud services. In *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*, pages 227–235. IEEE, 2013.

[71] Wei Li, Igor Santos, Flavia C Delicato, Paulo F Pires, Luci Pirmez, Wei Wei, Houbing Song, Albert Zomaya, and Samee Khan. System modelling and performance evaluation of a three-tier cloud of things. *Future Generation Computer Systems*, 70:104–125, 2017.

[72] Liang Tong, Yong Li, and Wei Gao. A hierarchical edge cloud architecture for mobile computing. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[73] Yi Hu, Feixiong Zhang, KK Ramakrishnan, and Dipankar Raychaudhuri. Geotopo: A pop-level topology generator for evaluation of future internet architectures. In *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, pages 90–99. IEEE, 2015.

[74] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 45(5):37–42, 2015.

[75] Wuyang Zhang, Yi Hu, Yanyong Zhang, and Dipankar Raychaudhuri. Segue: Quality of service aware edge cloud service migration. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 344–351. IEEE, 2016.

[76] Changsheng You, Kaibin Huang, Hyukjin Chae, and Byoung-Hoon Kim. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 16(3):1397–1411, 2016.

[77] Abdelhamied A Ateya, Anastasia Vybornova, Ruslan Kirichek, and Andrey Koucheryavy. Multilevel cloud based tactile internet system. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pages 105–110. IEEE, 2017.

[78] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. A hybrid edge-cloud architecture for reducing on-demand gaming latency. *Multimedia systems*, 20(5):503–519, 2014.

[79] Mathias Björkqvist, Lydia Y Chen, Marko Vukolić, and Xi Zhang. Minimizing retrieval latency for content cloud. In *2011 Proceedings IEEE INFOCOM*, pages 1080–1088. IEEE, 2011.

[80] Zhijing Qin, Grit Denker, Carlo Giannelli, Paolo Bellavista, and Nalini Venkatasubramanian. A software defined networking architecture for the internet-of-things. In *2014 IEEE network operations and management symposium (NOMS)*, pages 1–9. IEEE, 2014.

[81] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[82] Sumit Maheshwari, Dipankar Raychaudhuri, Ivan Seskar, and Francesco Bronzino. Scalability and performance evaluation of edge cloud systems for latency constrained applications. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 286–299. IEEE, 2018.

[83] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[84] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[85] Francesco Bronzino, Dipankar Raychaudhuri, and Ivan Seskar. Experiences with testbed evaluation of the mobilityfirst future internet architecture. In *Networks and Communications (EuCNC), 2015 European Conference on*, pages 507–511. IEEE, 2015.

[86] Open-access research testbed for next-generation wireless networks (orbit). https://www.orbit-lab.org/. Accessed: 2019-09-16.

[87] Yi Wang, Eric Keller, Brian Biskeborn, Jacobus van der Merwe, and Jennifer Rexford. Virtual routers on the move: live router migration as a network-management primitive. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 231–242. ACM, 2008.

[88] Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the internet impasse through virtualization. *Computer*, 38(4):34–41, 2005.

[89] Sumit Maheshwari, Shalini Choudhury, Ivan Seskar, and Dipankar Raychaudhuri. Traffic-aware dynamic container migration for real-time support in mobile edge clouds. In *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6. IEEE, 2018.

[90] Jinho Hwang, K K_ Ramakrishnan, and Timothy Wood. Netvm: High performance and flexible networking using virtualization on commodity platforms. *IEEE Transactions on Network and Service Management*, 12(1):34–47, 2015.

[91] Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, et al. Network virtualization in multi-tenant datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 203–216, 2014.

[92] Dmitry Drutskoy, Eric Keller, and Jennifer Rexford. Scalable network virtualization in software-defined networks. *IEEE Internet Computing*, 17(2):20–27, 2013.

[93] KK Ramakrishnan, Prashant Shenoy, and Jacobus Van der Merwe. Live data center migration across wans: a robust cooperative context aware approach. In *Proceedings of the 2007 SIGCOMM workshop on Internet network management*, pages 262–267. ACM, 2007.

[94] Timothy Wood, KK Ramakrishnan, Prashant Shenoy, and Jacobus Van der Merwe. Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines. In *ACM Sigplan Notices*, volume 46, pages 121–132. ACM, 2011.

[95] Vytautas Valancius, Nick Feamster, Jennifer Rexford, and Akihiro Nakao. Wide-area route control for distributed services. In *USENIX Annual Technical Conference*, 2010.

[96] Xiongqi Wu and James Griffioen. Supporting application-based route selection. In *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*, pages 1–8. IEEE, 2014.

[97] Andrew D Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. Participatory networking: An api for application control of sdns. In *ACM SIGCOMM computer communication review*, volume 43, pages 327–338. ACM, 2013.

[98] Francesco Bronzino, Sumit Maheshwari, Ivan Seskar, and Dipankar Raychaudhuri. Novn: named-object based virtual network architecture. In *Proceedings of the 20th International Conference on Distributed Computing and Networking*, pages 90–99. ACM, 2019.

[99] Francesco Bronzino, Shreyasee Mukherjee, and Dipankar Raychaudhuri. The named-object abstraction for realizing advanced mobility services in the future internet. In *Proceedings of the Workshop on Mobility in the Evolving Internet Architecture*, pages 37–42. ACM, 2017.

[100] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host Identity Protocol. RFC 5201, 2008.

[101] Dino Farinacci, Darrel Lewis, David Meyer, and Vince Fuller. The locator/ID separation protocol (LISP). RFC 6830, 2013.

[102] Michael Kowal, Dino Farinacci, and Parantap Lahiri. Lisp traffic engineering use-cases. Technical report, 2018.

[103] V Fuller, D Farinacci, D Meyer, and D Lewis. Locator/id separation protocol alternative logical topology (lisp+ alt). Technical report, 2013.

[104] Francesco Bronzino, Kiran Nagaraja, Ivan Seskar, and Dipankar Raychaudhuri. Network service abstractions for a mobility-centric future internet architecture. In *Proceedings of the eighth ACM international workshop on Mobility in the evolving internet architecture*, pages 5–10. ACM, 2013.

[105] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[106] Tam Vu, Akash Baid, Yanyong Zhang, Thu D Nguyen, Junichiro Fukuyama, Richard P Martin, and Dipankar Raychaudhuri. Dmap: A shared hosting scheme for dynamic identifier to locator mappings in the global internet. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 698–707. IEEE, 2012.

[107] Abhigyan Sharma, Xiaozheng Tie, Hardeep Uppal, Arun Venkataramani, David Westbrook, and Aditya Yadav. A global name service for a highly mobile internetwork. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 247–258. ACM, 2014.

[108] Yi Hu, Roy D Yates, and Dipankar Raychaudhuri. A hierarchically aggregated in-network global name resolution service for the mobile internet. Technical report, WINLAB TR 442, 2015.

[109] Yi Hu, Roy D Yates, and Dipankar Raychaudhuri. A hierarchically aggregated in-network global name resolution service for the mobile internet. *WINLAB: New-Brunswick, NJ, USA*, 2015.

[110] Sophie Y Qiu, Patrick D McDaniel, Fabian Monrose, and Aviel D Rubin. Characterizing address use structure and stability of origin advertisement in interdomain routing. In *11th IEEE Symposium on Computers and Communications (ISCC'06)*, pages 489–496. IEEE, 2006.

[111] Ratul Mahajan, David Wetherall, and Tom Anderson. A study of bgp origin as changes and partial connectivity. *Slide Presentation, University of Washington, Asta Networks,(ritual@ cs. washington. edu)(22 pages)*, 2001.

[112] Daniel Turull, Markus Hidell, and Peter Sjödin. Performance evaluation of openflow controllers for network virtualization. In *2014 IEEE 15th International Conference on High Performance Switching and Routing (HPSR)*, pages 50–56. IEEE, 2014.

[113] Yiannis Yiakoumis, Sachin Katti, and Nick McKeown. Neutral net neutrality. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 483–496. ACM, 2016.

[114] Kiyohide Nakauchi, Francesco Bronzino, Yozo Shoji, Ivan Seskar, and Dipankar Raychaudhuri. vmcn: virtual mobile cloud network for realizing scalable, real-time cyber physical systems. In *Proceedings of the 4th Workshop on Distributed Cloud Computing*, page 6. ACM, 2016.

[115] Samuel C Nelson, Gautam Bhanage, and Dipankar Raychaudhuri. Gstar: generalized storage-aware routing for mobilityfirst in the future mobile internet. In *Proceedings of the sixth international workshop on MobiArch*, pages 19–24. ACM, 2011.

[116] Mobilityfirst wiki. http://mobilityfirst.orbit-lab.org/.

[117] Ming Li, Devesh Agrawal, Deepak Ganesan, Arun Venkataramani, and Himanshu Agrawal. Block-switched networks: A new paradigm for wireless transport. In *NSDI*, volume 9, pages 423–436, 2009.

[118] Markus Feilner. *OpenVPN: Building and integrating virtual private networks*. Packt Publishing Ltd, 2006.

[119] YL Andersson, T Madsen, and AB Acreo. Provider Provisioned Virtual Private Network (VPN) Terminology. RFC 4026, 2005.

[120] Ignacio Castro, Juan Camilo Cardona, Sergey Gorinsky, and Pierre Francois. Remote peering: More peering without internet flattening. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 185–198. ACM, 2014.

[121] Xuxian Jiang and Dongyan Xu. Violin: Virtual internetworking on overlay infrastructure. *Parallel and Distributed Processing and Applications*, pages 937–946, 2005.

[122] Konstantinos Psounis. Active networks: Applications, security, safety, and architectures. *IEEE Communications Surveys*, 2(1):2–16, 1999.

[123] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 181–192. ACM, 2007.

[124] Jianli Pan, Subharthi Paul, Raj Jain, and Mic Bowman. Milsa: a mobility and multihoming supporting identifier locator split architecture for naming in the next generation internet. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–6. IEEE, 2008.

[125] Cedric Westphal. Challenges in networking to support augmented reality and virtual reality. *IEEE ICNC*, 2017.

[126] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[127] Francesco Bronzino, Sumit Maheshwari, Ivan Seskar, and Dipankar Raychaudhuri. Novn: named-object based virtual network architecture. In *Proceedings of the 20th International Conference on Distributed Computing and Networking*, pages 90–99. ACM, 2019.

[128] Zhou Zhao, Kai Hwang, and Jose Villeta. Game cloud design with virtualized cpu/gpu servers and initial performance results. In *Proceedings of the 3rd workshop on Scientific Cloud Computing*, pages 23–30. ACM, 2012.

[129] Sumit Maheshwari, Sudipta Mahapatra, Cheruvu Siva Kumar, and Kantubukta Vasu. A joint parametric prediction model for wireless internet traffic using hidden markov model. *Wireless networks*, 19(6):1171–1185, 2013.

[130] Lele Ma, Shanhe Yi, and Qun Li. Efficient service handoff across edge servers via docker container migration. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 11. ACM, 2017.

[131] Stephen J Vaughan-Nichols. New approach to virtualization is a lightweight. *Computer*, 39(11):12–14, 2006.

[132] Sumit Maheshwari, Saurabh Deochake, Ridip De, and Anish Grover. Comparative study of virtual machines and containers for devops developers. *arXiv preprint arXiv:1808.08192*, 2018.

[133] Claus Pahl and Brian Lee. Containers and clusters for edge cloud architectures–a technology review. In *2015 3rd international conference on future internet of things and cloud*, pages 379–386. IEEE, 2015.

[134] YC Tay, Kumar Gaurav, and Pavan Karkun. A performance comparison of containers and virtual machines in workload migration context. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 61–66. IEEE, 2017.

[135] The LXD container hypervisor. https://www.ubuntu.com/containers/lxd/ .

[136] Docker. https://www.docker.com/ .

[137] Shripad Nadgowda, Sahil Suneja, Nilton Bila, and Canturk Isci. Voyager: Complete container state migration. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2137–2142. IEEE, 2017.

[138] Andrew Machen, Shiqiang Wang, Kin K Leung, Bong Jun Ko, and Theodoros Salonidis. Live service migration in mobile edge clouds. *IEEE Wireless Communications*, 25(1):140–147, 2017.

[139] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K Leung. Dynamic service migration in mobile edge-clouds. In *2015 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2015.

[140] Chaima Ghribi, Makhlouf Hadji, and Djamal Zeghlache. Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 671–678. IEEE, 2013.

[141] CRIU. http://www.criu.org/ .

[142] Openalpr/openalpr. https://github.com/openalpr/openalpr/wiki/OpenALPR-Design .

[143] Piorkowski, M., et al. CRAWDAD dataset epfl/mobility. https://crawdad.org/epfl/mobility/20090224 .

[144] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[145] Shreyasee Mukherjee, Shravan Sriram, Tam Vu, and Dipankar Raychaudhuri. Eir: Edge-aware inter-domain routing protocol for the future mobile internet. *Computer Networks*, 127:13–30, 2017.

[146] ShareOn GitHub. https://github.com/sumitece87/shareon.

[147] David Geronimo, Antonio M Lopez, Angel D Sappa, and Thorsten Graf. Survey of pedestrian detection for advanced driver assistance systems. *IEEE transactions on pattern analysis and machine intelligence*, 32(7):1239–1258, 2010.

[148] Ryan W Wolcott and Ryan M Eustice. Visual localization within lidar maps for automated urban driving. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 176–183. IEEE, 2014.

[149] Mario Gerla, Eun-Kyu Lee, Giovanni Pau, and Uichin Lee. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 241–246. IEEE, 2014.

[150] Savaş Tay, P-A Blanche, R Voorakaranam, AV Tunç, W Lin, S Rokutanda, T Gu, D Flores, P Wang, G Li, et al. An updatable holographic three-dimensional display. *Nature*, 451(7179):694, 2008.

[151] Wenxiao Zhang, Bo Han, and Pan Hui. Jaguar: Low latency mobile augmented reality with flexible tracking. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 355–363. ACM, 2018.

[152] Zhanpeng Huang, Weikai Li, Pan Hui, and Christoph Peylo. Cloudridar: A cloud-based architecture for mobile augmented reality. In *Proceedings of the 2014 workshop on Mobile augmented reality and robotic technology-based systems*, pages 29–34. ACM, 2014.

[153] Wuyang Zhang, Sugang Li, Luyang Liu, Zhenhua Jia, Zhang Yanyong, and Dipankar Raychaudhuri. Heteroedge: Orchestration of real-time vision applications on heterogeneous edge clouds. In *INFOCOM, 2019 Proceedings IEEE*. IEEE, 2019.

[154] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[155] Ronald Azuma. Tracking requirements for augmented reality. *Communications of the ACM*, 36(7):50–51, 1993.

[156] Dipankar Raychaudhuri, Ivan Seskar, Max Ott, Sachin Ganu, Kishore Ramachandran, Haris Kremo, Robert Siracusa, Hang Liu, and Manpreet Singh. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 3, pages 1664–1669. IEEE, 2005.

[157] Alberto Ceselli, Marco Premoli, and Stefano Secci. Mobile edge cloud network design optimization. *IEEE/ACM Transactions on Networking (TON)*, 25(3):1818–1831, 2017.

[158] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[159] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997.

[160] Howard Rheingold. *Virtual reality: exploring the brave new technologies*. Simon & Schuster Adult Publishing Group, 1991.

[161] Swarun Kumar, Shyamnath Gollakota, and Dina Katabi. A cloud-assisted design for autonomous driving. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 41–46. ACM, 2012.

[162] Quanwen Zhu, Long Chen, Qingquan Li, Ming Li, Andreas Nüchter, and Jian Wang. 3d lidar point cloud based intersection recognition for autonomous driving. In *2012 IEEE Intelligent Vehicles Symposium*, pages 456–461. IEEE, 2012.

[163] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2017.

[164] Stefania Sardellitti, Gesualdo Scutari, and Sergio Barbarossa. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103, 2015.

[165] Konstantinos Samdanis, Xavier Costa-Perez, and Vincenzo Sciancalepore. From network sharing to multi-tenancy: The 5g network slice broker. *IEEE Communications Magazine*, 54(7):32–39, 2016.

[166] Peter Rost, Albert Banchs, Ignacio Berberana, Markus Breitbach, Mark Doll, Heinz Droste, Christian Mannweiler, Miguel A Puente, Konstantinos Samdanis, and Bessem Sayadi. Mobile network architecture evolution toward 5g. *IEEE Communications Magazine*, 54(5):84–91, 2016.

[167] Min Chen and Yixue Hao. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, 36(3):587–597, 2018.

[168] Allan Afuah. *Internet business models and strategies: Text and cases*. McGraw-Hill, Inc., 2002.

[169] David Gorodyansky. System and method for monetizing internet usage, November 23 2006. US Patent App. 11/471,247.

[170] Pankesh Patel, Ajith H Ranabahu, and Amit P Sheth. Service level agreement in cloud computing. 2009.

[171] C-H Chang, Pandelis Kourtessis, and JM Senior. Gpon service level agreement based dynamic bandwidth assignment protocol. *Electronics Letters*, 42(20):1173–1175, 2006.

[172] Karl Czajkowski, Ian Foster, Carl Kesselman, Volker Sander, and Steven Tuecke. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 153–183. Springer, 2002.

[173] Shreyasee Mukherjee, Shravan Sriram, Tam Vu, and Dipankar Raychaudhuri. Eir: Edge-aware inter-domain routing protocol for the future mobile internet. *Computer Networks*, 127:13–30, 2017.

[174] Linpack. https://www.netlib.org/linpack/. Accessed: 2019-09-16.

[175] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[176] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[177] Jian Guo, Fangming Liu, Dan Zeng, John CS Lui, and Hai Jin. A cooperative game based allocation for sharing data center networks. In *2013 Proceedings IEEE INFOCOM*, pages 2139–2147. IEEE, 2013.

[178] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma. Performance analysis of load balancing algorithms. *World Academy of Science, Engineering and Technology*, 38(3):269–272, 2008.

[179] Xiaodan Wang, Christopher Olston, Anish Das Sarma, and Randal Burns. Coscan: cooperative scan sharing in the cloud. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 11. ACM, 2011.

[180] G Serban and MC Pintea. Heuristics and learning approaches for solving the travling salesman problem. *Studia Universitatis Babes-Bolyai Informatica*, 49(2):27–36, 2004.

[181] Home page - bgpview. https://bgpview.io/. Accessed: 2019-09-16.

[182] Hurricane electric bgp toolkit. https://bgp.he.net/. Accessed: 2019-09-16.

[183] Github – finding lane lines using python and opencv. https://github.com/naokishibuya/car-finding-lane-lines/. Accessed: 2019-07-01.

[184] Disco. https://github.com/sumitece87/disco. Accessed: 2019-09-16.

[185] Sumit Maheshwari, Prasad Netalkar, and Dipankar Raychaudhuri. Disco: Distributed control plane architecture for resource sharing in heterogeneous mobile edge cloud scenarios. In *40th IEEE International Conference on Distributed Computing Systems (ICDCS), Singapore*, 2020 (to appear).