Author/s:
Roy, D;Rao, AS;Alpcan, T;Das, G;Palaniswami, M

Title:
Achieving AI-Enabled Robust End-to-End Quality of Experience Over Backhaul Radio Access Networks

Date:
2022-09-01

Citation:
Roy, D., Rao, A. S., Alpcan, T., Das, G. & Palaniswami, M. (2022). Achieving AI-Enabled Robust End-to-End Quality of Experience Over Backhaul Radio Access Networks. IEEE Transactions on Cognitive Communications and Networking, 8 (3), pp.1468-1481. https://doi.org/10.1109/TCCN.2022.3177516.

Achieving AI-enabled Robust End-to-End Quality of Experience over Backhaul Radio Access Networks

Dibbendu Roy, Aravinda S. Rao, *Senior Member, IEEE,* Tansu Alpcan, *Senior Member, IEEE,* Goutam Das, and Marimuthu Palaniswami, *Fellow, IEEE*

***Abstract*—Emerging applications such as Augmented Reality, the Internet of Vehicles and Remote Surgery require both computing and networking functions working in harmony. The End-to-end (E2E) quality of experience (QoE) for these applications depends on the synchronous allocation of networking and computing resources. However, the relationship between the resources and the E2E QoE outcomes is typically stochastic and non-linear. In order to make efficient resource allocation decisions, it is essential to model these relationships. This article presents a novel machine-learning based approach to learn these relationships and concurrently orchestrate both resources for this purpose. The machine learning models further help make robust allocation decisions regarding stochastic variations and simplify robust optimization to a conventional constrained optimization. When resources are insufficient to accommodate all application requirements, our framework supports executing some of the applications with minimal degradation (graceful degradation) of E2E QoE. We also show how we can implement the learning and optimization methods in a distributed fashion by the Software-Defined Network (SDN) and Kubernetes technologies. Our results show that deep learning-based modelling achieves E2E QoE with approximately 99.8% accuracy, and our robust joint-optimization technique allocates resources efficiently when compared to existing differential services alternatives.**

***Index Terms*—E2E QoE, Network Slicing, Kubernetes, SDN, O-RAN.**

I. Introduction

New applications and use cases are largely stimulated by modern networking architectures and solutions for 5G and beyond. The International Telecommunication Union has classified 5G mobile network services into three categories: (1) Enhanced Mobile Broadband (bandwidth-intensive like Virtual Reality), (2) Ultra-reliable and Low-latency Communications (highly delay-sensitive and (3) reliable like automated driving), and Massive Machine Type Communications (high connection density like IoT and Industry 4.0) [1]. In 6G, there might be further granularity involved in defining these categories [2] based on applications. In addition, 6G aims at integrating intelligence in the network, implying that the network should interpret user requirements and take suitable decisions to satisfy them. For example, Australian telecommunication service provider Telstra promises at delivering programmable networks as per user needs [3].

User requirements are typically built upon classical metrics such as delay, throughput, jitter, and reliability. The aforementioned new applications require satisfying end-to-end (E2E) Quality of Experience (QoE) based on these metrics. However, the relationship between the E2E QoE parameters and the resources is non-linear (typically non-convex) and stochastic [4]. It also challenges service providers to allocate resources to satisfy the E2E QoE requirements of emerging applications. Most of the available works either assume that the relationship between E2E QoE and resources is simplistic or that the users are aware of the required resources for achieving the desired E2E QoE. However, these assumptions might not be realistic given the complex and dynamic nature of these applications.

Even though we talk of networks, these applications require services from both networking and computing. Modern networking technologies such as *software defined networks (SDN)* [5] and and computing technologies like *Kubernets (K8)* [6] and *dockers* [7] enable virtualization of networks and applications, which allows each application to have its own network and computing resources (refer Section II). Thus, service providers must jointly design and adapt their networking and computing resources to satisfy the diverse E2E QoE requirements. Although a plethora of research work is available for satisfying QoE metrics separately in network and computing domains, a comprehensive study on the interaction of the two while considering the relevant technologies in these paradigms is missing (refer Section II).

This paper presents a modern approach to joint computing and networking resource orchestration. Building upon the technological aspects involved in characterizing E2E QoE, we develop a sequential machine learning and optimization-based framework to meet E2E QoE requirements, primarily focusing on robustness aspects.

The contributions of the paper can be summarized as:

- We address the problem of E2E QoE satisfaction in emerging applications with the help of modern networking and computing technologies. To the best of our knowledge, this is the first work that combines both networking and computing aspects concerning E2E QoE while considering SDN and Kubernetes as orchestrators of the network and computing sites.
- We consider the backhaul segment of a radio access network with bursty traffic arrivals. In this segment, the applications can be virtualized at the base band unit

This research was supported partially by the Australian Government through the Australian Research Council's Discovery Projects funding scheme (project DP190102828).

D. Roy, A. S. Rao, T. Alpcan and M. Palaniswami are with the Department of Electrical and Electronic Engineering, The University of Melbourne, Parkville, Victoria - 3010, Australia (e-mail: dibbendu.roy@student.unimelb.edu.au, aravinda.rao@unimelb.edu.au, tansu.alpcan@unimelb.edu.au, palani@unimelb.edu.au).

G. Das is now with the G.S. Sanyal School of Telecommunications, Indian Institute of Technology Kharagpur, West Bengal - 721302, India. (e-mail: gdas@gssst.iitkgp.ac.in).

(BBU) and served by the core and edge server. Most of the existing works assume that the applications are aware of the resources required to satisfy the QoE which is not the case in practice. In this paper, we propose to use AI to model these relationships with the help of deep learning followed by formulating an optimization. The learning models helps to simplify a robust optimization problem to a conventional constrained optimization by directly learning the worst case possibilities.

- Typically, most of the available white-papers and standard documents [8], [9] segregate the network domains into RAN, transport network (TN), and core network (CN). Each of these have their individual orchestrators while a centralized supervisor tries to manage the overall E2E QoE. The common approach to do this is to allocate achievable QoE goals for each domain and try to maintain them independently with the help of controllers. This makes the problem simpler and easy to implement. We follow this approach initially and show how our framework can be implemented as a digital twin [10] using emulation platform. Although for the majority of the paper, the fronthaul or radio access has not been considered, we state and show how the same can be incorporated in the proposed model.
- When network or computing resources are constrained, it is desirable to have minimal service degradation depending on service level agreements. In these situations, services can be optimally reconfigured or gracefully degraded with the help of the presented optimization framework.
- Implementing a digital twin might not be an efficient solution to many service providers as it comes at cost of additional processing and security threat. The proposed approach can be applied on a real network setting if SDN controller and Kubernetes can work independently with minimal coordination. Thus, it is important to learn and optimize in a distributed manner. We present the challenges in the distributed learning scenario and show how the problem can be tackled with help of primal decomposition technique. This implementation comes at cost of additional storage for training over all examples rather than the worst ones.
- Our results show the efficacy of using deep learning towards achieving uRLLC requirements. Compared to existing methods, it can achieve uRLLC targets of reliability of more than 99%. Further, we show the effect of using joint-optimization instead of the existing prioritization standards at network and computing sites. While joint-optimization can satisfy the uRLLC requirements, the same is not the case for other approaches. We also demonstrate the utility of graceful degradation through our approach.

The rest of the paper is organized as follows: Section II contains relevant background and gaps in literature leading to Section III which discusses the problem, model and methodology for our learning and optimization approach. We discuss the learning methodology followed by joint robust optimization techniques in section IV-B and IV. Finally, we present the relevant emulation and simulation results in Section VII followed by concluding remarks in Section VIII.

II. Background and Literature Review

Modern networking technologies move away from traditional solutions, which require specialized networking hardware and are challenging to maintain and configure. It is envisioned that 6G networks will be developed over SDN [5] which separate the control and data planes at the network devices. The notion of separation simplifies hardware requirements for network devices and increases flexibility as we can program them from a centralized controller. We can implement networking operations such as routing, bandwidth allocation, and security using virtual machines (VMs) on a general server which is commonly known as *network function virtualization (NFV)* [5]. From a purely computing perspective, there have been some critical advances in virtualization technologies as well. The concept of having VMs with hardware-level segregation of resources has been replaced with lightweight middleware applications called *containers* [7]. Similar to VM, each container has its own set of resources managed at the OS level with the help of software such as *dockers*. For container deployments over multiple servers, production-level container orchestration software, such as *Kubernetes (K8)* [11], manages and maintains the containers. With the help of these emerging technologies, service providers can satisfy E2E QoE requirements by implementing a logical allocation of the network and computing infrastructure, also known as *network slicing*. Based on these technologies, next-generation access networks are moving towards Open Radio Access Network (O-RAN) [12], which aims to provide RAN functionalities on open hardware and software solutions.

We now provide a brief review of relevant research literature in this area. There are several proposals for resource allocation in the case of networks and cloud computing for achieving QoE [5], [13]–[28]. Broadly, we can classify these works as those involving computing (Kubernetes based) [23]–[27] and the ones involving networking (SDN based) [5], [13]–[22]. The slicing approaches formulated over SDN typically find/reconfigure routes and decide on bandwidth allocations along the routes. The authors of [13], [14] evaluate their performance by calculating the time taken to create a network slice. Some existing works are listed in [28] which mainly focus on re-configuring slices using AI-based techniques. These works adapt to changing slice requirements. However, while designing their slice configuration or re-configuration algorithms, they assume that the applications know the resource requirements. This assumption is quite restrictive as applications or users would be only concerned about E2E QoE, and the mapping between E2E QoE and resources is often complex, dynamic, and stochastic.

The authors in [23]–[27] mostly deal with Kubernetes settings and container deployment strategies. However, they do not incorporate the associated networking delays. Thus, E2E QoE has not been investigated in these works.

Although most of the papers mentioned above do not consider the roles of SDN and Kubernetes in their models,

[16], [18] consider network, computing and E2E QoE. In these works, the authors solve the problem of VM placement based on server and network loads which turns out to be a Mixed-Integer linear optimization problem (MILP). The authors consider a simple linear relationship of E2E delays concerning network loads which is not the case in reality due to randomness and non-linear behaviour of delay concerning loads, as evident from elementary queuing theory. Further, they do not consider the effect on E2E delays due to the resulting placement strategies.

III. Problem, Model, and Methodology

A. Problem Statement

From our previous discussions in Sections I and II, it is clear that there have been very few works that consider E2E delays from both networks and computing perspectives. In existing works, applications/tasks have their bandwidth and processing requirements, and they place these at servers where they can achieve load balancing. Thus, the papers mentioned above have modeled these problems as assignment problems requiring binary decisions resulting in mixed-integer problems. Apart from the assignment, modern computing technologies such as Dockers and Kubernetes allow deploying applications with different resources. The decision implies that the requisite computing resources must be assigned to ensure E2E QoE. Users should only care about their experience while the resource allocation decisions (networking and computing) are intelligently decided as envisioned in 6G networks. This paper proposes techniques to model E2E QoE requirements and subsequently perform joint multi-resource allocation of networking and computing resources to satisfy the desired E2E QoE requirements.

In this paper, we address the following problems:

- How to model E2E QoE with respect to bandwidth and processing as resource variables?
- How to obtain robust optimal decisions regarding resource variables?
- How to handle resource constrained situations?
- How to implement our approach as a digital twin using emulation environments?
- How to implement the allocation decisions with independent network and computing orchestrators?

Before we proceed to our methodology, we present the relevant notations and system model for our work.

B. Model and Notation

We consider a Radio Access Network (RAN) where users are connected to a Baseband Unit (BBU) through a remote radio head (RRH) and a router (see Fig. 1). The BBU is equipped with Kubernetes or Docker system to manage containers (applications) that run on them. An SDN controller can configure the router, and users host different applications with QoE requirements served at the BBU. The SDN and Kubernetes create E2E slices for the backhaul portion of the network. Later we show how this can be extended for the fronthaul part as well. The SDN and Kubernetes controllers

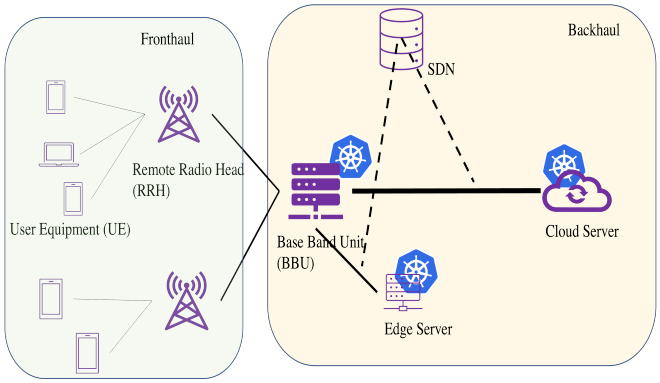


Fig. 1: A 5G Radio Access Network

can communicate and share statistics related to QoE metrics required to decide the resources for a slice.

Users host different applications, with different QoE requirements. A slice comprises applications having similar QoE requirements. In our model, we consider that an application sends packets related to the application to be executed at an edge or cloud server. The packets carry the required data for executing the job. For example, an application might be an algorithm to sort an array where packets contain array data. In such applications, the type of application and the amount of data it sends determine the degree of processing required at the server. Thus, each application can have its traffic generation rates and processing demands.

Further, depending on the type of protocol employed at MAC and Transport layers, the delays experienced at the network sites may vary. An E2E user experience can thus be characterized by the delays experienced to obtain the result of an application request. Also, there might be situations when packets get lost either at the network (considering a UDP based protocol for real-time applications) or at the server (due to overload or out of memory). In such cases, the user does not get a response from the server leading to an undesirable experience. To model this effect, we define E2E throughput as the percentage of successful requests. Thus, we use the two metrics, namely E2E delay and E2E throughput, for characterizing user experience.

1) Slice: We define a slice to be a set of applications having similar QoE requirements. It is up to the service providers to decide upon how they would want to associate applications to slices. Our objective is to provide a generic model that can be extended as required. Let $\mathbb{A} \in \{a_1, \ldots, a_N\}$ denote the set of classes or slices with different QoE requirements. Each slice a_i demands a QoE depending on its requirements given by the tuple $q_i = (\tau_i, \rho_i)$, where τ_i denotes the desired E2E delay, ρ_i denotes the desired E2E throughput. As per requirements, q_i may have several criteria. E2E delay comprises of all relevant delays such as access, propagation, queuing, transmission and processing in the network and the computing sites. Thus, a slice a_i is uniquely associated and is defined by the requirement q_i.

2) Network Model: Typically, corresponding to a slice, a service provider would decide the bandwidth at the network side, the corresponding processor allocation at the edge/cloud

TABLE I: Notations used in this work.

Notation	Description
$\mathbb{A}$	Set of Applications
$G = (V, E)$	Physical network graph. V is set of vertices and E is the set of edges/links. Routing decisions rely on link capacities.
$B(e)$	Capacity of link $e \in E$
θ_i	Random variable to model uncertainty in E2E QoE due to stochastic variations for example arrivals
QoE Parameters	
τ_i	E2E delay requirement for application $a_i \in \mathbb{A}$
ρ_i^N	Throughput requirement from network for application $a_i \in \mathbb{A}$
ρ_i^S	Throughput requirement from server for application $a_i \in \mathbb{A}$
r_i	Probability figure requirement that $a_i \in \mathbb{A}$ does not fail either due to network or server problems
Slicing Decision Variables	
f_i^e	Flow for $a_i \in \mathbb{A}$ along edge $e \in E$ (implemented by SDN/Network Controller)
ϕ_i^c	Fraction of CPU core c allocated at the server side for $a_i \in \mathbb{A}$ (implemented by Kubernetes)
Functions in Optimization	
$D_i(.)$	Delay constraint function
$T_i(.)$	Throughput constraint function
$u_i(.)$	Utility function (can be average delay, sum throughput, cost etc.)
$\psi_i(.)$	Penalty function for graceful degradation of constraint to be relaxed

server and routing decisions if the network comprises of several routes or paths. To characterize a network the general approach is to define a graph.

Let $G = (V, E)$ denote a graph where V denotes the set of vertices and E denotes the set of edges. Apart from the network, let us consider a server with C cores. A slicing decision may be modeled as a tuple (f_i^e, ϕ_i^c) $\forall e \in E$ and $\forall c \in C$, where $f_i^e \in [0, 1]$ denotes the normalized flow rate for slice a_i along edge e. The normalization is performed over the bandwidth capacity of each link. This decision on f_i^e is taken by an SDN controller or network hypervisor like FlowVisor [29].

$\phi_i^c \in [0, 1]$ denotes the fraction of the processing for core c at a server. This modeling allows us to capture parallelizable applications as well, and its realistic implementations are feasible due to the advent of dockers and Kubernetes based server systems [6], [7]. Like CPU allocation decisions, Kubernetes can also decide on the amount of RAM to be allocated for a slice. This can also be introduced as a decision variable along with ϕ_i^c. However, we found that changing the amounts of RAM do not significantly affect the E2E delays and E2E throughput. In our experiments and this paper, we assume that the server has sufficient RAM to execute an application. Thus, we have ignored the decision variable for RAM in the paper. However, there might be situations where RAM might be important while handling large matrices or such operations. In such cases, the model can be suitably extended. The variables for each e and c may be collected and stacked as a vector. We denote this by bold letters. However, for the considered RAN network, the decisions boil down to scalar variables due to single link and consideration of non-parallel applications in the paper. In a general network setting, the slicing decisions would also involve routing decisions. The use of normalized flow variables indicate a probabilistic routing strategy. However, for our simulations we employed a network with a single route from BBU to the cloud or server as is the practice for most Telecom service providers like Telstra.

Each of the two major E2E QoE parameters, namely delay and throughput can be captured via implicit functions of the form:

$$\text{E2E Delay: } D_i(\boldsymbol{f_i^e}, \boldsymbol{\phi_i^c}; \boldsymbol{\theta_i}) \tag{1}$$

$$\text{E2E Throughput: } T_i(\boldsymbol{f_i^e}, \boldsymbol{\phi_i^c}; \boldsymbol{\theta_i}) \tag{2}$$

Here, E2E throughput is determined by the number of successful requests. A request is successful if it is successfully transferred by the network and subsequently processed at the server. Other QoE parameters can also be included to extend our model. Although we write implicit functions for mapping the QoE constraints to decision variables, it is essential to note that these functions may also depend on one or more stochastic parameters. These parameters are denoted by $\boldsymbol{\theta_i}$ which can be associated with a range space $\boldsymbol{\Theta_i}$. A simple example of such a parameter is the number of customers/subscribers for a given slice. Since in this paper, we consider a RAN network, the vector parameters are ignored henceforth and we use scalar notations.

3) Extension to GPUs: Modern machine learning applications require GPU computation and hence it might be of interest to investigate how the presented model can accommodate for the same. As mentioned in [30], [31], support for GPUs have been included in container based management systems. Thus, machine learning applications deployed as containers with GPU allocations, can be used, similar to the CPU use-case as shown in this paper. Depending on the parallelizability of the developed application, the number of GPU cores used in the application would vary. Our model already captures multi-core allocation policies as ϕ_i^c involves decisions on a per core basis and hence would be able to handle GPU based allocations as well.

C. Methodology

As discussed in Section I, this paper focuses on satisfying QoE requirements over the 5G network and beyond. The QoE performances depend on allocated computing and networking resources. It is well understood that the resource allocation with QoE constraints can be cast as a constrained optimization problem as has been primarily explored in the literature (refer Section II). However, the QoE parameters may exhibit complex dependencies with those of the resources, and hence such problems have been difficult to solve in general [19]–[21].

In this work, we try to create network slices in two phases. In the first phase, we exploit the power of deep learning to learn these complex dependencies. In the second phase, we solve an optimization problem to obtain allocation/slicing decisions. When we say a slice, it implies creating a network

slice and corresponding allocation at the server-side. Our approach for solving the problem can be depicted as shown in Fig. 2. When a service provider receives a new slice request (QoE requirements), a service provider may implement the digital twin with help of available emulation platforms for SDN and Kubernetes (refer Section VI). It then probes the network and server according to agreed-upon service level agreements. By varying the networking and computing resources, one can obtain the desired QoE metrics. A deep neural network is very efficient in learning any arbitrary function [32]. Fig. 2 shows a conceptual overview of how our approach may be implemented in a digital twin [10]. Using the digital twin, the E2E delays and E2E throughput of clients may be emulated for various application requests. Then a neural network is trained to capture the complex relationship between the resource parameters and QoE metrics.

Once these dependencies are obtained, one can solve an optimization problem to obtain robust network slices. In case that resources are not enough to satisfy the QoE requirements, some slices might have to be reconfigured so that critical slices satisfy their QoE requirements at the cost of relaxed QoE satisfaction of other slices. We term this as graceful degradation of services which is desirable in the case of resource-constrained scenarios. In this work, we assume that the service level agreements are negotiated between the slices and service providers, allowing the service providers to probe network and servers suitably. These decisions might change with newly negotiated or renegotiated agreements. The loop in our proposed approach Fig. 2 captures this effect.

IV. Sequential Learning and Joint Multi-Resource Allocation

We adopt learning based robust optimisation approach for multi-resource allocation. Specifically, we present the constraints, objective, and robust optimization approach for joint allocation of network and computing resources. Apart from the E2E QoE constraints, the capacity constraints for communication links and processing of the server should be taken into consideration for joint optimization.

A. E2E QoE Constraints

The QoE functions discussed before are restricted by the following bounds:

$$D_i(f_i^e, \phi_i^c; \theta_{\mathbf{i}}) \leq \tau_i \quad \forall\ i, \theta_i \tag{3}$$

$$T_i(f_i^e, \phi_i^c; \theta_{\mathbf{i}}) \geq \rho_i \quad \forall\ i, \theta_i \tag{4}$$

Here, $D_i(f_i^e, \phi_i^c; \theta_{\mathbf{i}})$ and $T_i(f_i^e, \phi_i^c; \theta_{\mathbf{i}})$ are the learned functions discussed in Section IV-B

B. Learning the E2E QoE Model

It is well known that by Universal Approximation Theorem [32], neural networks act as universal function approximators [33] and hence can be used to learn or approximate arbitrary dependencies (see Fig. 2). Based on the nature of the constraints, one may apply a suitable neural network to approximate its nature. To account for the stochastic nature of network state and server, service provider uses a Monte-Carlo strategy (by using different random seeds to generate traffic) and obtain the required statistics. For example, for a given bandwidth and processing configuration, one can obtain the histogram for E2E delays experienced by the packets. Once the data regarding the QoE metrics are collected, a neural network is trained as shown in Fig. 2, for a digital twin based implementation.

C. Link Capacity

Given a link $e \in E$ from the network graph G, the capacity of the link is given by $B(e)$. The bandwidth allocated to slices routed through this link should not exceed this capacity. Since f_i^e denotes the normalized flow rate (normalized with respect to $B(e)$) for slice a_i, we have

$$\sum_{i|a_i \in \mathbb{A}} f_i^e \leq 1 \quad \forall e \in G \tag{5}$$

D. Server Capacity

The fraction of processing for a given core c at a server should not exceed the maximum processing capability of the core. We consider a core as a unit processing element and all cores are identical. Thus, we must have

$$\sum_{i|a_i \in \mathbb{A}} \phi_i^c \leq 1 \quad \forall\ c \in C \tag{6}$$

Here, C denotes the set of cores of a CPU in a server.

E. Objective Function

Network Slicing can be performed depending on many desired objectives such as sum throughput maximization, delay minimization, minimization of delay violation, load balancing etc. Without loss of any generality, we consider general utility functions which may be mapped to any of the aforementioned objectives (similar to well-known network utility maximization). Let $u_i(f_i^e, \phi_i^c)$ denote the utility obtained on executing application a_i over an edge network. Then, a generic objective might be of the form:

$$\sum_{i|a_i \in \mathbb{A}} u_i(f_i^e, \phi_i^c; \theta_i) \tag{7}$$

In this paper, we use throughput as the utility. $u_i(f_i^e, \phi_i^c; \theta_i) = T_i(f_i^e, \phi_i^c; \theta_i)$. Another possible utility function is the sum of resources which is to be minimized.

F. Robust Optimization using Machine Learning

Before dealing with the distributed version of the problem, we state the robust centralized version of the problem. This can be directly implemented in a digital twin based implementation. A robust design is one which works for all possible variations of the parameters in the parameter space. Thus, by robustness we strive to design slices with worst possible parameter combinations. Hence, we may pose the robust optimization problem as a max-min problem, where

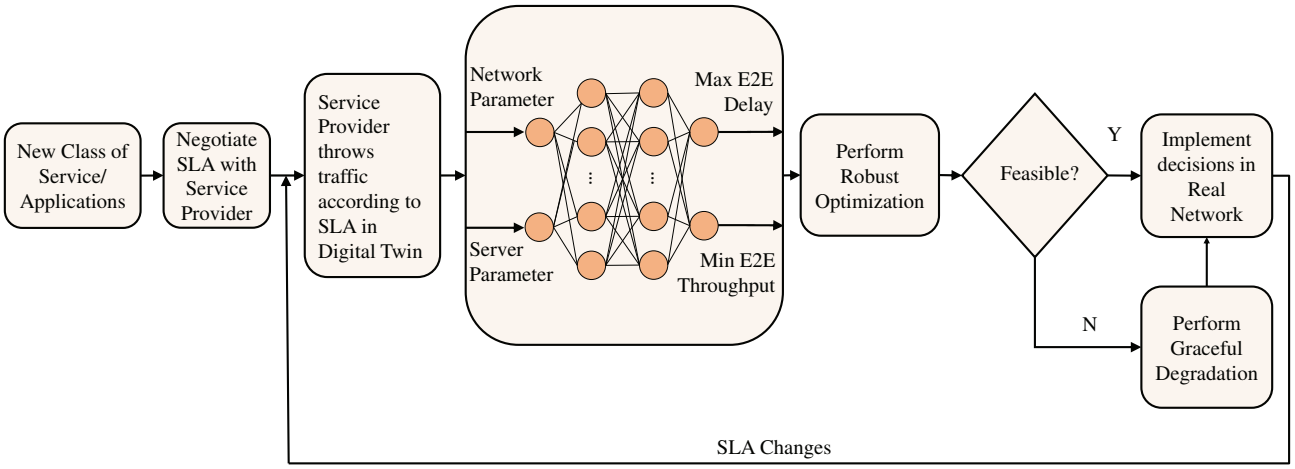


Fig. 2: Conceptual View of Sequential Learning and Joint Multi-Resource Allocation for E2E QoE using Digital Twin

we maximize our objective subject to worst possible effect of the parameters, in this case θ_i on the problem [34].

Using (3),(4),(5),(6) and (7), we may state the robust Optimization problem as:

$$\max_{(f_i^e,\phi_i^c)} \min_{\theta_i \in \theta_i} \sum_{i|a_i \in \mathbb{A}} u_i(f_i^e, \phi_i^c; \theta_i) \tag{8a}$$

$$\text{s.t. } D_i(f_i^e, \phi_i^c; \theta_\mathbf{i}) \leq \tau_i \quad \forall i, \theta_i \tag{8b}$$

$$T_i(f_i^e, \phi_i^c; \theta_i) \geq \rho_i \quad \forall i, \theta_i \tag{8c}$$

$$\sum_{i|a_i \in \mathbb{A}} f_i^e \leq 1 \quad \forall e \in G \tag{8d}$$

$$\sum_{i|a_i \in \mathbb{A}} \phi_i^c \leq 1 \quad \forall\ c \in C \tag{8e}$$

It is important to note that solving a constrained robust optimization problem such as (8) is computationally complex. Since we use deep learning to learn the E2E QoS metrics, we may also use the same methodology to directly learn the minimized utility function over θ_i. For example, a robust optimization strategy for throughput would be to maximize the throughput while considering the worst possible effect of θ_i on throughput. In general, it would be difficult to characterize the worst possible effect. However, with help of learning we can directly learn the worst possible throughput using neural networks. As discussed in Section IV-B, we vary the seeds for generating traffic and capture the stochastic variations in delays and throughput at the network and computing sites. For each point, we take the maximum delay and minimum throughput values for worst case modeling and feed to the neural network for training purposes. Thus, the robust optimization can be converted to a normal optimization as:

$$\max_{(f_i^e,\phi_i^c)} \sum_{i|a_i \in \mathbb{A}} u_i(f_i^e, \phi_i^c) \tag{9a}$$

$$\text{s.t. } D_i(f_i^e, \phi_i^c) \leq \tau_i \quad \forall i \tag{9b}$$

$$T_i(f_i^e, \phi_i^c) \geq \rho_i \quad \forall i \tag{9c}$$

$$\sum_{i|a_i \in \mathbb{A}} f_i^e \leq 1 \quad \forall e \in G \tag{9d}$$

$$\sum_{i|a_i \in \mathbb{A}} \phi_i^c \leq 1 \quad \forall\ c \in C \tag{9e}$$

In (9), and foregoing discussions, we omit the stochastic parameter θ_i as its effect is captured using neural network. In Appendix of the paper, we show how the sequential learning and robust optimization problem may be solved in a distributed manner at cost of additional storage and learning. For optimization, any non-linear optimizer might be used. We used the sequential quadratic program as our functions were twice continuously differentiable and at each iteration, we can also obtain the lagrange multipliers required for the distributed implementation.

G. Graceful Degradation

The problem mentioned above may be infeasible for resource-constrained scenarios. In such situations, it is desirable that the service provider be interested in relaxing the constraints such that the most prioritized slices do not violate their constraints. The notion of priority may be conceived as a mapping of the requested QoE parameters. Based on the priority, the service provider may impose appropriate penalties, thereby allowing them to gracefully degrade (relax low priority constraints to make slicing feasible) the service. We define $\psi_i(D_i(.) - \tau_i, T_i(.) - \rho_i; \theta_i)$ as the associated penalty function for application class a_i. For application classes which require strict QoE satisfaction, a penalty of zero is assigned i.e., $\psi_i(.) = 0$, if a_i has strict QoE requirement. Let $\mathbb{B}$ and $\mathbb{C}$ denote two sets of application classes such that those in $\mathbb{B}$ must satisfy their QoE in strict sense while those in $\mathbb{C}$ may be degraded.

Then, the service provider may solve the following optimization to obtain a feasible set of QoE constraints that may be satisfied by the original optimization.

$$\min_{(f_i^e,\phi_i^c)} \sum_{i|a_i\in\mathbb{C}} \psi_i\Big(D_i(.)-\tau_i, T_i(.)-\rho_i\Big) - \sum_{i|a_i\in\mathbb{A}} u_i(f_i^e,\phi_i^c) \tag{10a}$$

$$\text{s.t. } D_i(f_i^e,\phi_i^c) \le \tau_i \quad \forall\ i\in\mathbb{B} \tag{10b}$$

$$T_i(f_i^e,\phi_i^c) \ge \rho_i \quad \forall\ i\in\mathbb{B} \tag{10c}$$

$$\sum_{i|a_i\in\mathbb{A}} f_i^e \le 1 \quad \forall e\in G \tag{10d}$$

$$\sum_{i|a_i\in\mathbb{A}} \phi_i^c \le 1 \quad \forall\ c\in C \tag{10e}$$

In solving this problem, the optimal slicing and feasible QoE constraints may be obtained based on the priority settings similar to the previous case.

V. Distributed Networking and Computing Implementation

As discussed in Section I, for service providers without the capability of implementing a digital twin, it is desirable that the SDN and Kubernetes achieve E2E QoE in a distributed manner. We observe that the decision variables of the two entities are different. The SDN has the responsibility to decide f_i^e while Kubernetes decides ϕ_i^c. We turn our attention to constraints.

It is evident that the E2E delay and throughput functions depend on both variables. Due to additive nature of delays, the E2E delay can be expressed as sum of delay functions of network and computing site respectively. Note that E2E throughput is determined by number of successful requests. A request is successful if it is successfully transferred by the network and subsequently processed at the server. The two events of success at network and at server are independent and hence their probabilities can be multiplied. To convert this into a sum, we may use a logarithm operation over the throughput function. Further, the randomness associated with the random variable θ_i can be accounted using equations (11),(12).

$$\forall\theta_i,\ D_i(f_i^e,\phi_i^c;\theta_i) = D_i^N(f_i^e;\theta_i) + D_i^S(\phi_i^c;\theta_i) \tag{11}$$

$$\forall\theta_i,\ T_i(f_i^e,\phi_i^c;\theta_i) = T_i^N(f_i^e;\theta_i) \times T_i^S(\phi_i^c;\theta_i) \tag{12}$$

As shown in Fig. 3a, the SDN obtains the network delays experienced by the packets when packets flow within the network via the routers. One can implement a network monitoring application [35] to collect relevant delay and drop statistics. Using a network hypervisor like Flowvisor, the network can be sliced with different bandwidth configurations [36] for the packets corresponding to a specific application. Similar to the SDN, Kubernetes [37] can vary the computing power allocated to a container (application) and calculate the delays and drops (crashes/out of memory) to execute the container, if any. To account for the stochastic nature of network state and server, service provider uses a Monte-Carlo strategy (by using different random seeds to generate traffic) and obtain the required statistics. For example, for a given bandwidth and processing configuration, one can obtain the histogram for E2E delays experienced by the packets. Once the data regarding the QoE metrics are collected by the SDN and Kubernetes, they may train individual neural networks as shown in Fig. 3a. The E2E delay and throughput may be computed using (11) and (12). Although in case of centralized robust solution, we could learn the worst case delays directly, the same cannot be done for a distributed approach. This is due to the fact that maximum or minimum are non-linear operators. Thus, for each realization of θ_i, the delays and throughput values are computed by SDN and Kubernetes and fed into the neural networks at corresponding sites. We use primal decomposition technique for distributed optimization [38]. The overall algorithm is shown in Fig. 3b. The decision variables of the two entities - SDN and Kubernetes are different, namely f_i^e and ϕ_i^c respectively. SDN may have its own objective (say $u_i^N(f_i^e)$) and Kubernetes may have another one (say $u_i^S(\phi_i^c)$). These objectives may or may not depend on θ_i. In this section, we present the simpler case where these are independent of θ_i and present the more general version in the next subsection of the paper.

A. Distributed Robust Optimization without Stochastic Objective

As presented in (11) and (12), the E2E delay and E2E throughput are separable as for a given θ_i. For decentralized implementation, it is important that we decompose the functions as sums instead of a product. This can be easily achieved for E2E throughput by taking a logarithm on both sides of (12):

$$\forall\theta_i,\ \ \log T_i(f_i^e,\phi_i^c,\theta_i) = \log T_i^N(f_i^e,\theta_i) + \log T_i^S(\phi_i^c,\theta_i) \tag{13}$$

We introduce variables τ_i^N and ρ_i^N such that:

$$D_i^N(f_i^e,\theta_i) \le \tau_i^N \text{ and } D_i^S(\phi_i^c,\theta_i) \le \tau_i - \tau_i^N$$

$$\log T_i^N(f_i^e,\theta_i) \le \log\rho_i^N \text{ and } \log T_i^S(\phi_i^c,\theta_i) \le \log\rho_i - \log\rho_i^N$$

Thus for a given $\theta_i, \tau_i^N, \rho_i^N$, the SDN and Kubernetes can solve their own optimization problems as:

P(N)[SDN]:

$$\max_{f_i^e} \sum_{i|a_i\in\mathbb{A}} u_i^N(f_i^e)$$

$$\text{s.t. } D_i^N(f_i^e,\theta_i) \le \tau_i^N$$

$$\log T_i^N(f_i^e,\theta_i) \le \log\rho_i^N$$

$$\sum_{i|a_i\in\mathbb{A}(e)} f_i^e \le 1 \quad \forall e\in G$$

P(S)[Kubernetes]:

$$\max_{\phi_i^c} \sum_{i|a_i\in\mathbb{A}} u_i^S(\phi_i)$$

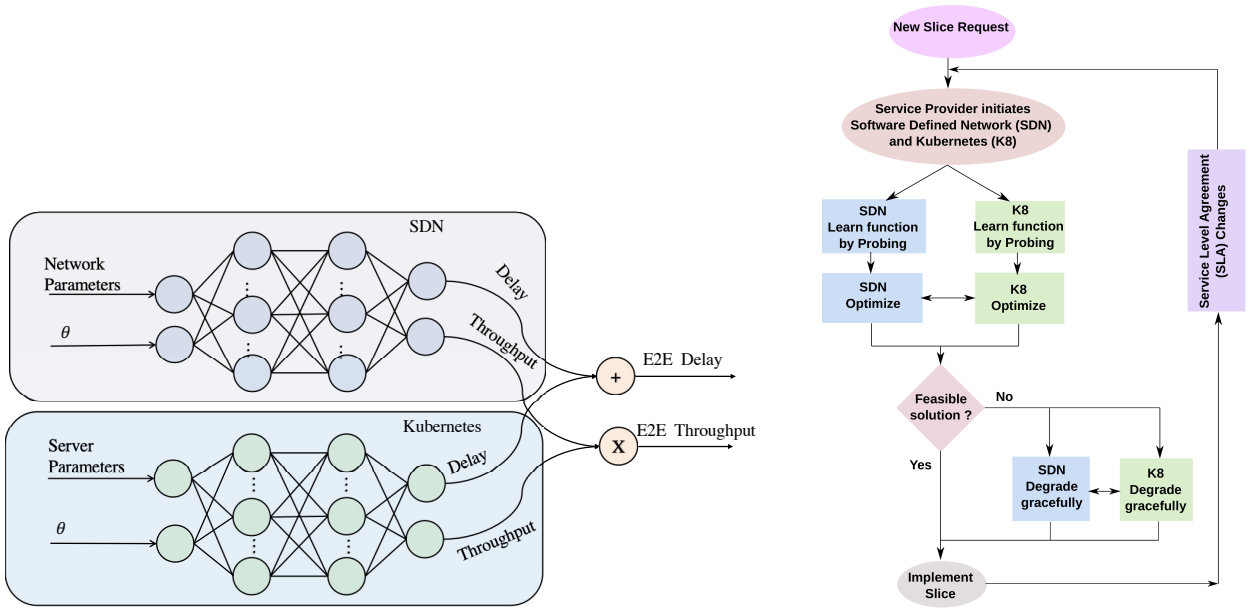
$$\text{s.t. } D_i^S(\phi_i^c,\theta_i) \le \tau_i - \tau_i^N$$

$$\log T_i^S(\phi_i,\theta_i) \le \log\rho_i - \log\rho_i^N$$

$$\sum_{i|a_i\in\mathbb{A}} \phi_i^c \le 1 \quad \forall\ c\in C$$

On solving these, the SDN and Kubernetes obtains the optimal f_i^e, ϕ_i^c for a given $(\tau_i^N, \rho_i^N, \theta_i)$. We may express the optimal value of the utilities by $N(\tau_i^N,\rho_i^N,\theta_i)$ and $S(\tau_i^N,\rho_i^N,\theta_i)$. Finally the objective is the maximize the sum of the obtained functions which is often termed as the master problem [38]:

$$\max_{(\tau_i^N,\rho_i^N,\theta_i)} N(\tau_i^N,\rho_i^N,\theta_i) + S(\tau_i^N,\rho_i^N,\theta_i) \tag{16}$$



(a) Learning at SDN and Kubernetes with separate neural networks

(b) Overview of Distributed learning and optimization technique.

Fig. 3: Distributed Implementation of Sequential Learning and Robust Optimization in SDN and Kubernetes

Note that $\tau_i^N \in [0, \tau_i]$, $\rho_i^N \in [0, \rho_i]$ and $\theta_i \in \theta_i$. Let us denote the domains for τ_i and ρ_i by $\mathbb{T}$ and $\mathbb{P}$. Let $\Pi_{\mathbb{C}}(\mathbf{x})$ denote the projection of the point $\mathbf{x}$ over set $\mathbb{C}$. The sub-gradient update equations for $(\tau_i^N, \rho_i^N, \theta_i)$ are:

$$\tau_i^N := \Pi_{\mathbb{T}}\Big(\tau_i^N + \alpha(\lambda_\tau^N - \lambda_\tau^S)\Big) \tag{17a}$$

$$\rho_i^N := \Pi_{\mathbb{P}}\Big(\rho_i^N + \alpha(\lambda_\rho^S - \lambda_\rho^N)\Big) \tag{17b}$$

$$\theta_i := \Pi_{\theta_i}\Big(\theta_i + \alpha(g_{\theta_i}^N + g_{\theta_i}^S)\Big) \tag{17c}$$

$\lambda_\tau^N, \lambda_\tau^S, \lambda_\rho^S, \lambda_\rho^N$ denote the Lagrange multipliers associated with the constraints w.r.t to τ, ρ at SDN and Kubernetes respectively. $g_{\theta_i}^N$, $g_{\theta_i}^S$ denotes the sub-gradients w.r.t θ_i of the formed Lagrangians at SDN and Kubernetes respectively. Although, we wrote the generalized projection operators, the operators are quite straightforward for the sets described above. For, $\tau_i^N \in [0, \tau_i]$ the projection operator is simply the minimum of the obtained value and τ_i or else zero in case the obtained value is negative. The same is the case for ρ_i^N. For θ_i which models the arrival process, the obtained value must be non-negative at each step.

Algorithm 1 shows the steps for distributed optimization. As starting points, we choose $\tau_i^N = \frac{\tau_i}{2}$, $\rho_i^N = \frac{\rho_i}{2}$, $\theta_i = \bar{\theta}_i$. $\bar{\theta}_i$ denotes the average arrival rate. We have well defined problems at SDN and Kubernetes sites which are solved by them in parallel. We use sequential quadratic programming which also provides us with the Lagrange multipliers. These are to be exchanged among SDN and Kubernetes after each iteration. The algorithm stops if the gradients (difference of lagrange multipliers) are less than some predefined small number ϵ. It is to be noted that the obtained solution is locally optimal in case of generalized functions and global optimal is achieved under convexity assumptions.

Algorithm 1: Algorithm for Distributed Implementation

Input: ϵ, $\tau_i^N = \frac{\tau_i}{2}$, $\rho_i^N = \frac{\rho_i}{2}$,$\theta_i = \bar{\theta}_i$ Objectives and Constraints of P(N) and P(S)

Output: $f_i^{*e}, \phi_i^{*c} \quad \forall\ i$

while *True* **do**

- ***@SDN**: SDN solves P(N) to obtain f_i^e and optimal Lagrange multipliers $\lambda_\tau^N, \lambda_\rho^N, g_{\theta_i}^N$;*
- ***@Kubernetes:** Kubernetes solves P(S) to obtain ϕ_i^c and optimal lagrange multipliers $\lambda_\tau^S, \lambda_\rho^S, g_{\theta_i}^S$;*
- Exchange Multipliers and sub-gradients;
- Update for Master Problem using (22);
- Stopping Criteria: $|\lambda_\tau^N - \lambda_\tau^S|$ & $|\lambda_\rho^N - \lambda_\rho^S|$ & $(g_{\theta_i}^N + g_{\theta_i}^S) < \epsilon$;

end

B. Distributed Robust Optimization with Stochastic Objective

In this section, we discuss the solution to the generalized distributed robust optimization problem where the objective function depends on the stochastic parameter θ_i. The optimization problem to be solved in distributed manner is that of (8), where the objective function contains the stochastic parameter θ_i. For example, utility is throughput, as considered in the paper. However, it is essential that SDN and Kubernetes design their own objectives which are coupled by the variable θ_i i.e.

$$u_i(f_i^e, \phi_i^c, \theta_i) = u_i^N(f_i^e, \theta_i) + u_i^S(\phi_i^c, \theta_i) \tag{18}$$

For a given, θ_i, the problem is again separable, with separate objectives for SDN and Kubernetes. Since the aim is to perform robust optimization i.e. maximizing the utility

over the worst possible effect of θ_i i.e. minimized over θ_i. However, on fixing θ_i, the only optimization variables left are f_i^e and ϕ_i^c and hence the sub-problems solved by SDN and Kubernetes are still the same.

P(N)[SDN]:

$$\max_{f_i^e} \sum_{i|a_i\in\mathbb{A}} u_i^N(f_i^e,\theta_i)$$
$$\text{s.t. } D_i^N(f_i^e,\theta_i) \le \tau_i^N$$
$$\log T_i^N(f_i^e,\theta_i) \le \log \rho_i^N$$
$$\sum_{i|a_i\in\mathbb{A}(e)} f_i^e \le 1 \quad \forall e \in G$$

P(S)[Kubernetes]:

$$\max_{\phi_i^c} \sum_{i|a_i\in\mathbb{A}} u_i^S(\phi_i,\theta_i)$$
$$\text{s.t. } D_i^S(\phi_i^c,\theta_i) \le \tau_i - \tau_i^N$$
$$\log T_i^S(\phi_i,\theta_i) \le \log \rho_i - \log \rho_i^N$$
$$\sum_{i|a_i\in\mathbb{A}} \phi_i^c \le 1 \quad \forall\, c \in C$$

On solving these, the SDN and Kubernetes obtains the optimal f_i^e, ϕ_i^c for a given $(\tau_i^N, \rho_i^N, \theta_i)$. We may express the optimal value of the utilities by $N(\tau_i^N, \rho_i^N, \theta_i)$ and $S(\tau_i^N, \rho_i^N, \theta_i)$ as before. Hoever, the master problem now changes to:

$$\max_{(\tau_i^N,\rho_i^N)\in\mathbb{T}\times\mathbb{P}} \min_{\theta_i\in\theta_i} \quad N(\tau_i^N,\rho_i^N,\theta_i) + S(\tau_i^N,\rho_i^N,\theta_i) \quad (21)$$

This problem is equivalent to finding saddle-points or commonly known as the saddle-point problem [39]. The sub-gradient update equations for $(\tau_i^N, \rho_i^N, \theta_i)$ are:

$$\tau_i^N := \Pi_{\mathbb{T}}\Big(\tau_i^N + \alpha(\lambda_\tau^N - \lambda_\tau^S)\Big) \quad (22a)$$

$$\rho_i^N := \Pi_{\mathbb{P}}\Big(\rho_i^N + \alpha(\lambda_\rho^S - \lambda_\rho^N)\Big) \quad (22b)$$

$$\theta_i := \Pi_{\theta_i}\Big(\theta_i - \alpha(g_{\theta_i}^N + g_{\theta_i}^S)\Big) \quad (22c)$$

The rest of the steps are as shown in Algorithm 1.

VI. Implementation of Digital Twin

To demonstrate a digital twin implementation, we used an emulator named ComNetsEmu developed at Granelli Lab [36]. The emulator is developed over the traditional SDN emulator MININET where each host has docker installed for container management [36]. In order to mimic Kubernetes, ComNetsEmu uses a docker-in-docker based concept that allows managing resources within a docker host. We present emulation results with three clients. Due to the limitations of Mininet, links could not support data rates greater than 1 Gbps.

We implement a radio access network setup as shown in Fig.5. Three clients were connected to an edge server via an OpenFlow switch which an SDN controller controls. The propagation delays of the links are distributed uniformly between 5 μs and 25 μs (1-5 Km). Each client sends UDP packets to the server. The packets contain array data, and the server finds Fast Fourier Transforms of the data and returns the result. To differentiate between applications of different processing requirements, we consider an application (App1) which computes the transform 1000 times for a request while the other (App2) does the exact 10000 times, respectively. We set the delay constraints for the two applications to be 2ms and 6ms for the emulation setup.

The E2E delays are collected and learnt using a $2\times16\times32\times8\times1$ neural network. Fig. 4a shows the performance of the network. The collected values are plotted over the learnt function. The figure shows that the delays exhibit non-linear nature, as mentioned before. Fig. 4b shows the performance of joint-optimization over the learnt E2E delays. The optimal CPU and bandwidth allocations returned all sent requests, and hence the throughput for both the slices were 100%. However, as can be seen from Fig.4b, App1 violates delay constraints by 2.3% while App2 does so by 6%. The small percentage of violation is due to the combined effect of learning inefficiency as well as the non-convex nature of the optimization problem.

VII. Experimental Results

Although the emulator is quite pragmatic, large scale emulations (connecting more than ten clients) could not be supported on a laptop. To circumvent this, we created a simulator in OmNet++ based on our experience with emulation and cloud computing simulators like CloudSim [40].

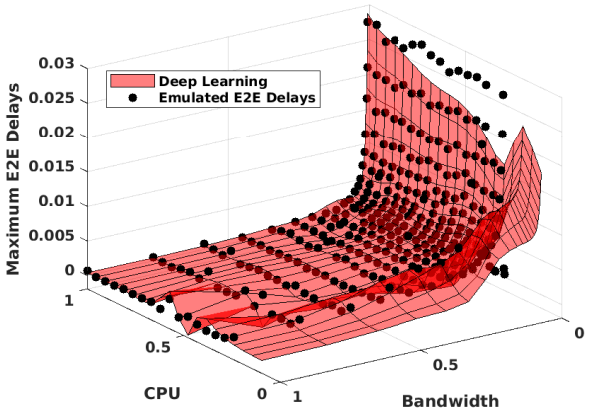
We conduct multiple experiments using simulation environments. These experimental scenarios are motivated by a suggestion from an actual network operator (Telstra) in the deployment of 5G networks in Australia.

A. Simulation Results

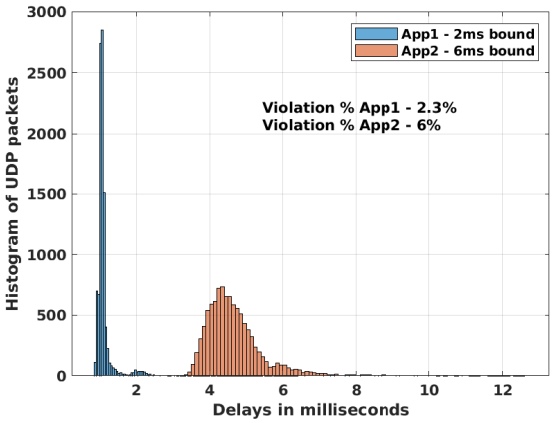
TABLE II: Parameters considered in simulations

Parameter	Value
(Delay, Throughput) requirements of App1	(1ms, 90%)
(Delay, Throughput) requirements of App2	(5ms, 95%)
#Cores/Server	2
Core Speed @Edge	3e8 MIPS
Processing required for App1	5e4 MI
Processing required for App2	8e4 MI
Core Speed @Cloud	3e9 MI
#Active Users	10
Packet generation rate	200/second
Packet Arrival Process	Bursty Pareto H = 8
Packet Size (Bytes)	$\sim U(20, 65535)$
Propagation Delay to edge (1-5 Km)	$\sim U(5\mu s, 25\mu s)$
Propagation Delay from edge to cloud	0.5 ms
Bandwidth between subscriber and edge (100 Km)	1 Gbps each
Bandwidth between edge and cloud	100 Gbps

We present the large scale simulation studies (clients>10). To demonstrate the performance of our slicing approach for the physical networking scenario described in Fig. 5, we



(a) Learnt E2E delays using Neural Network on Emulation Platform



(b) Histogram of E2E delays of requests for Slice1 and Slice2. Delay violations show for two application.

Fig. 4: Figures showing implementation of sequential learning and optimization for E2E QoE satisfaction in Emulation.

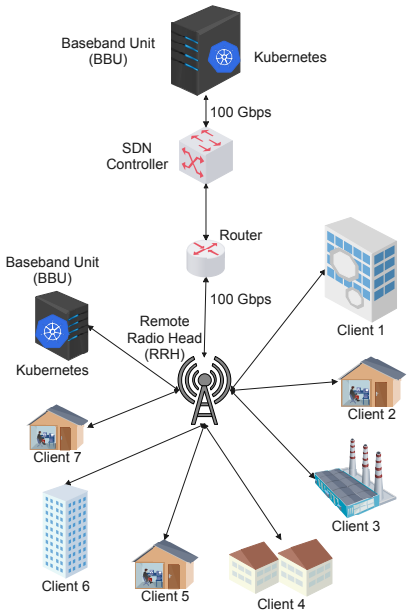


Fig. 5: Simulation setting for E2E QoE constraints over RAN.

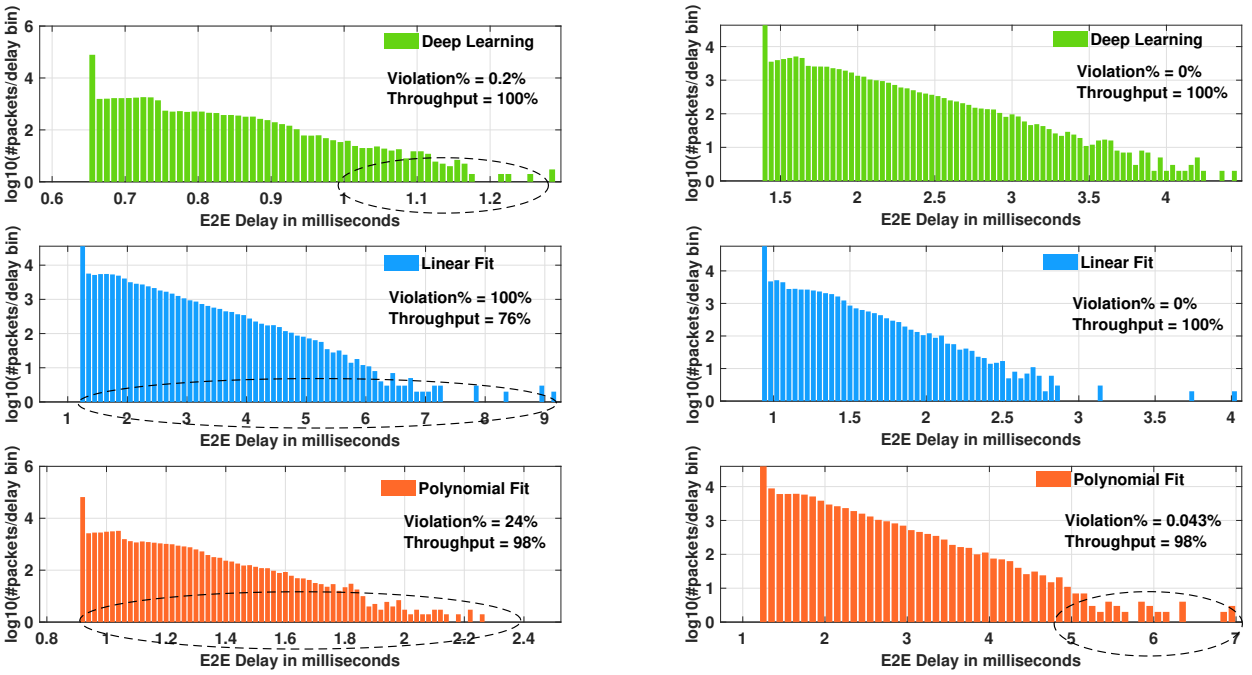
implement the described network in OmNet++ with ten subscribers, each requesting 200 packets per second. The traffic arrival process is bursty generated using an on-off Pareto distribution. We have two types of slices corresponding to applications App1 and App2, respectively. For App1, each packet sends a request to the server to execute an operation with an average of 5×10^4 Million instructions (MI) to be executed at an edge server that operates at a speed of 3×10^8 million instructions per second (MIPS). Similarly, for App2, we have a higher computing requirement of 8×10^4 MI. The end to end delay constraints for the two slices are set at 1ms and 5ms, respectively, while the throughput requirements are set at 95% and 97%, respectively. It is easily understood that the slice of App1 cannot be implemented at the Cloud since the round trip delay is already 1ms, and no delay < 1 ms can be achieved at Cloud. In order to satisfy these requirements, we create a network slice out of the total bandwidth 100 Gbps and total computing at the edge server. The bandwidth slice is implemented by the SDN, while Kubernetes implements the slice at the edge server.

B. Efficacy of Deep Learning

To evaluate the performance of our slicing method, we measure the end to end delays and throughput on receiving the output at the client-side. By throughput, we mean the percentage of requests that was returned to the client successfully. We plot the histogram of packets with respect to end to end delays experienced by them. It is essential to observe that App2 resembles a heavy computation application compared to App1, which has moderate computing and networking requirements.

To compare our method, we consider the works of [16], [17] where a linear relationship was used to estimate the delays. However, it is essential to note that these works addressed the problem of placement of applications at servers and did not deal with the processor allocation while considering the modern deployment strategies in Dockers and Kubernetes. We also compare the obtained slicing decisions with a polynomial fit instead of a deep neural network used for our approach. The polynomial used is of degree 5 in one dimension and degree 4 in the other.

The linear relationship proposed in the literature does not yield a feasible solution for App1 while we are able to obtain feasible solutions for App2 (satisfies the constraints) whose performances can be observed from the throughput percentages and end to end delay histograms. As seen from Fig. 6a and Fig. 6b, our approach shows that the number of packets violating the end to end delay constraints are significantly less compared to polynomial estimates. This is presumably due to the efficacy of deep neural network as an universal function approximator. We observe that in our approach none of the packets requesting App2 violate the delay constraints while some of those of App1 do violate.



(a) Performance of Deep learning approach vs Linear [16], [17] and Polynomial Fit for App1

(b) Performance of Deep learning approach vs Linear [16], [17] and Polynomial Fit for App2

Fig. 6: Figures showing the effectiveness of sequential deep learning over other learning approaches. Linear approach [16], [17] is skewed towards one application while polynomial approach performs relatively better. Deep learning approach improves performance by a significant amount.

This may be due to prediction errors and is marginal as shown in Fig. 6a. Deep learning approach promises to achieve uRLLC requirements of satisfying strict delay constraints for 99.99% of time. While this is exactly the case for App2, for App1 the same happens for 99.2% of times. Linear Fit approach goes towards either extreme (infeasible allocations in one and allocates all resources to other) due to inefficient approximation of the E2E QoE functions.

C. Joint Optimization vs Differentiated Service

To demonstrate the effectiveness of the joint-optimization technique proposed in the paper, we compare the same with prioritization based service, which is the standard way of dealing with delay constrained services [6], [41]. In our simulations, we preempt the prioritized App over other Apps requests either at the networking site or server site. Fig. 7a shows the E2E delay performance of received packets for 10^5 packets of App1 (1ms bound). As can be seen from Fig. 7a, the number of packets that violate the delay constraint is only 0.2% as compared to the network prioritized and computing prioritized approaches. This is due to the nature of the application. The result also suggests that the App1 is sensitive to both network and computing. Only the joint-optimization technique can effectively handle its QoE requirements. Hence in either differentiated methods, we observe that the obtained delays violate the required bounds.

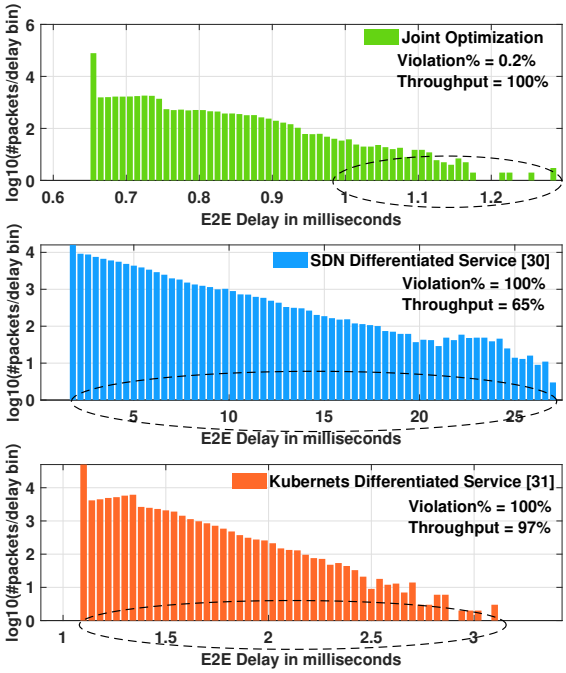
Fig. 7b shows the E2E delay performance of received packets for 10^5 packets of App2 (5ms bound). As can be seen from Fig. 7a, none of packets violate the delay constraint compared to the network prioritized approach. This is again due to the nature of the application. The result suggests that the App2 is sensitive to computing. In this case, assigning more computing power helps in achieving the required delay constraint.

D. Graceful Degradation

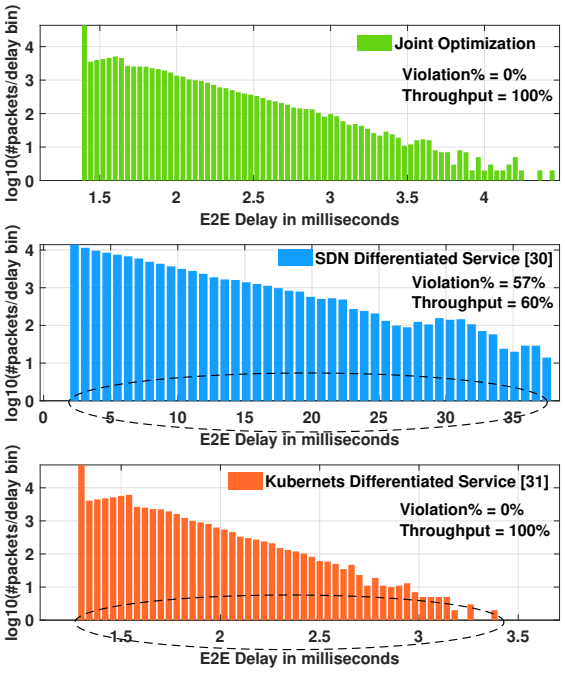
We introduce a new App - App3 with 8e4 MI and a delay constraint of $8ms$, which returns an infeasible solution (constraint satisfaction is not possible). In such cases, our approach proposes using the graceful degradation framework, which aims to find minimum possible relaxations on the requirements. In this regard, we want App2 and App3 to strictly satisfy its constraints while imposing a penalty on the deviation of the App1 delay constraint in our objective. As we observe from Fig. 8, the optimization finds the feasible delay constraint to be 3ms. However, we observe that the new slicing decisions have increased the number of packets violating delay for App3. This might be due to involved errors in learning the functions.

E. Extension for fronthaul RAN

The aforementioned techniques were discussed with respect to backhaul RAN wherein, the applications are virtualized at



(a) Performance of Joint Optimization vs Differentiated service for App1.



(b) Performance of Joint Optimization vs Differentiated service for App2.

Fig. 7: Figures showing the effectiveness of sequential deep learning and joint-optimization approach over Differentiated Services. App1 is given priority at queues in the network and server sides respectively. App1 is both network and processing hungry. App2 is given priority at queues in the network and server sides respectively. App2 is more processor hungry as compared to App1 while both compete for network.

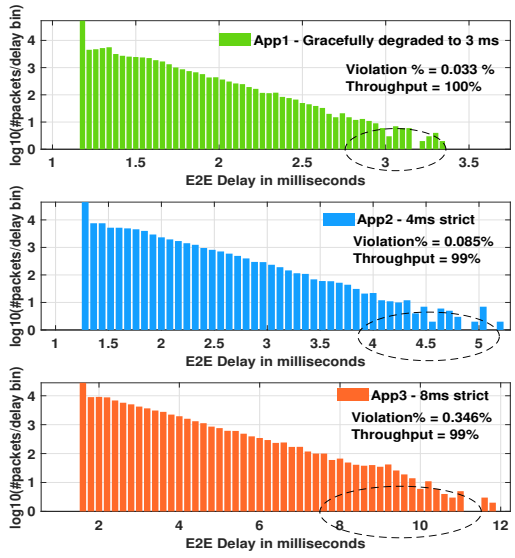


Fig. 8: App3 is introduced. App2 and App3 have strict delay constraints. App1 is gracefully degraded to 3 ms.

the BBU and implemented at the server. As shown, a digital twin based implementation over mininet was plausible as this part of the network is wired and is easy to virtualize on a desktop or laptop.

The typical approach to slicing is to segregate the network into access network (AN), transport network (TN) and core network (CN) and define slices individually in these domains [8], [9]. A centralized orchestrator, then has the responsibility to manage these independently such that the overall QoE are met. We used minimization of resources as the utility function in this case. As pointed out in [42], RAN slicing (at the wireless or access side) is achieved by employing a virtual resource block allocation by a RAN scheduler or controller. The physical resource blocks correspond to time-frequency elements allocated to users in RAN. Once virtual resource blocks are decided, the scheduler maps them to physical resource blocks. Assuming that such a mapping is always possible, we can employ our model to control the RAN slicing as well. In this case, the RAN controller (at BBU) decides on the virtual resource blocks which becomes another input to our neural network (it can come under network parameters as it is also a decision variable that configures the network). Let us denote this variable as $R_i \in [0, 1]$ where it represents the percentage of time-frequency grid allocated to slice a_i. We simulated the channel in OMNet++ and used Rayleigh fading along with a path-loss model to model the long-term fading effects. The modulation scheme used for each symbol of a slice depends on the channel condition and is decided as per channel quality index. Fig. 9 and Fig. 10 show the

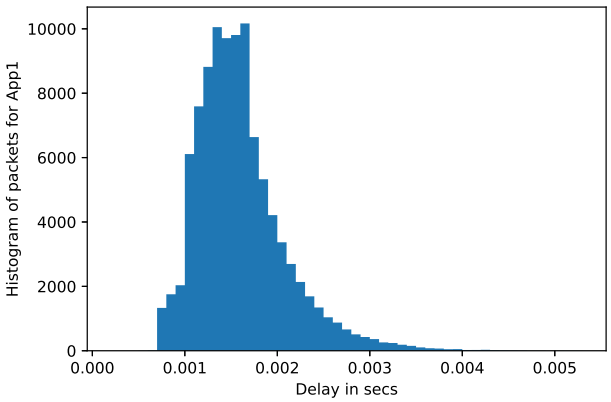


Fig. 9: E2E Delays with wireless for App1

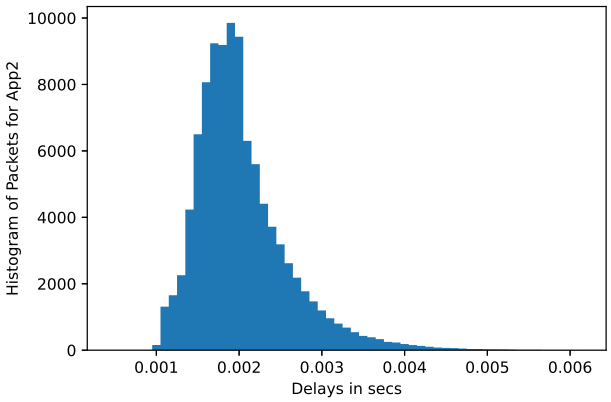


Fig. 10: E2E Delays with wireless for App2

E2E delay histograms as a result of wireless allocations. Apps corresponding to Slice1 (App1) is gracefully degraded in this case and violates the delay bound of $1ms$ and can only achieve the desired delay bound guarantees for $4ms$. It was found that resource block arrangement was saturated with a sharing of around 63% and 37% for the two slices. The CPU allocation was also saturated with ratio around 33% and 67% while the bandwidth through wired network was not saturated.

VIII. Conclusion

In this paper, we use deep learning to learn the relationship between resources and E2E QoE metrics for slicing in radio access networks (RAN). Deep learning proves to be highly efficient in modelling non-linear relationships compared to polynomial fitting approaches and linear approximations used in literature. In addition, learning helps in simplifying a robust optimization problem to a conventional optimization one. We also show how the learning and optimization can be implemented in a distributed manner on network and server controllers at cost of additional memory. Further, we show how we can make resource allocation decisions in resource-constrained scenarios by gracefully degrading a slice. However, learning approach is not free from errors and its difficult to ensure bounds rather than practical implementations. In this regard, conservative design approach may be employed i.e. to find allocations for stricter bounds than what is agreed upon. In this work, we only considered E2E delays and throughput as QoE metrics. We can also consider other relevant QoE metrics, such as the jitter. Large scale studies over servers and networks with such frameworks can be carried out. Theoretical extensions could involve finding bounds on the robustness of thus found optimal solutions.

Acknowledgment

The authors would like to thank Mr. Akilan Wick from Telstra Australia for his useful inputs regarding the Australian 5G deployment scenario.

References

[1] J. Sachs, G. Wikstrom, T. Dudda, R. Baldemair, and K. Kittichokechai, "5g radio network design for ultra-reliable low-latency communication," *IEEE network*, vol. 32, no. 2, pp. 24–31, 2018.

[2] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J. A. Zhang, "The roadmap to 6g: Ai empowered wireless networks," *IEEE Communications Magazine*, vol. 57, no. 8, pp. 84–90, 2019.

[3] "Telstra programmable network by telstra enterprise." [Online]. Available: https://www.telstra.com.au/business-enterprise/products/networks/sdn/telstra-programmable-network

[4] C. She, Y. Duan, G. Zhao, T. Q. Quek, Y. Li, and B. Vucetic, "Cross-layer design for mission-critical iot in mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 9360–9374, 2019.

[5] P.-K. Chartsias, A. Amiras, I. Plevrakis, I. Samaras, K. Katsaros, D. Kritharidis, E. Trouva, I. Angelopoulos, A. Kourtis, M. S. Siddiqui *et al.*, "Sdn/nfv-based end to end network slicing for 5g multi-tenant networks," in *2017 European Conference on Networks and Communications (EuCNC)*. IEEE, 2017, pp. 1–5.

[6] "Pod priority and preemption," Jul 2021. [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/

[7] "Build docker kubernetes-ready applications on your desktop." [Online]. Available: https://www.docker.com/products/kubernetes

[8] "200420_samsung network slicing," https://images.samsung.com/is/content/samsung/assets/global/business/networks/insights/white-paper/network-slicing/200420_Samsung_Network_Slicing_Final.pdf, (Accessed on 04/02/2022).

[9] "Ts 128 530 - v15.2.0 - 5g; management and orchestration; concepts, use cases and requirements (3gpp ts 28.530 version 15.2.0 release 15)," https://www.etsi.org/deliver/etsi_ts/128500_128599/128530/15.02.00_60/ts_128530v150200p.pdf, (Accessed on 04/02/2022).

[10] "Cheat sheet: What is digital twin?" https://www.ibm.com/blogs/internet-of-things/iot-cheat-sheet-digital-twin/, (Accessed on 11/14/2021).

[11] "Production-grade container orchestration." [Online]. Available: https://kubernetes.io/

[12] S. Niknam, A. Roy, H. S. Dhillon, S. Singh, R. Banerji, J. H. Reed, N. Saxena, and S. Yoon, "Intelligent o-ran for beyond 5g and 6g wireless networks," *arXiv preprint arXiv:2005.08374*, 2020.

[13] Z. Shu and T. Taleb, "A novel qos framework for network slicing in 5g and beyond networks based on sdn and nfv," *IEEE Network*, vol. 34, no. 3, pp. 256–263, 2020.

[14] D. L. C. Dutra, M. Bagaa, T. Taleb, and K. Samdanis, "Ensuring end-to-end qos based on multi-paths routing using sdn technology," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[15] A. T. Oliveira, B. J. C. Martins, M. F. Moreno, A. B. Vieira, A. T. A. Gomes, and A. Ziviani, "Sdn-based architecture for providing qos to high performance distributed applications," in *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2018, pp. 00 602–00 607.

[16] D. Sattar and A. Matrawy, "Optimal slice allocation in 5g core networks," *IEEE Networking Letters*, vol. 1, no. 2, pp. 48–51, 2019.

[17] D. Dietrich, C. Papagianni, P. Papadimitriou, and J. S. Baras, "Network function placement on virtualized cellular cores," in *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*. IEEE, 2017, pp. 259–266.

[18] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, 2017.

[19] U. Zehra and M. A. Shah, "A survey on resource allocation in software defined networks (sdn)," in *2017 23rd International Conference on Automation and Computing (ICAC)*, 2017, pp. 1–6.

[20] Y. L. Lee, J. Loo, T. C. Chuah, and L.-C. Wang, "Dynamic network slicing for multitenant heterogeneous cloud radio access networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 4, pp. 2146–2161, 2018.

[21] A. Baumgartner, T. Bauschert, A. M. Koster, and V. S. Reddy, "Optimisation models for robust and survivable network slice design: A comparative analysis," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–7.

[22] M. Jiang, M. Condoluci, and T. Mahmoodi, "Network slicing in 5g: An auction-based model," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[23] F. Rossi, V. Cardellini, and F. L. Presti, "Hierarchical scaling of microservices in kubernetes," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 2020, pp. 28–37.

[24] S. Hirai, T. Tojo, S. Seto, and S. Yasukawa, "Automated provisioning of cloud-native network functions in multi-cloud environments," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 1–3.

[25] F. A. Wiranata, W. Shalannanda, R. Mulyawan, and T. Adiono, "Automation of virtualized 5g infrastructure using mosaic 5g operator over kubernetes supporting network slicing," in *2020 14th International Conference on Telecommunication Systems, Services, and Applications (TSSA*. IEEE, 2020, pp. 1–5.

[26] R. Figueiredo and K. Subratie, "Edgevpn. io: Open-source virtual private network for seamless edge computing with kubernetes," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 190–192.

[27] M. Gawel and K. Zielinski, "Analysis and evaluation of kubernetes based nfv management and orchestration," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 511–513.

[28] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, and Y.-C. Liang, "Network slice reconfiguration by exploiting deep reinforcement learning with large action space," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2197–2211, 2020.

[29] "Github - opennetworkinglab/flowvisor: Flowvisor - a network hypervisor," https://github.com/opennetworkinglab/flowvisor, (Accessed on 09/17/2021).

[30] "Runtime options with memory, cpus, and gpus — docker documentation," https://docs.docker.com/config/containers/resource_constraints/, (Accessed on 09/17/2021).

[31] "Schedule gpus — kubernetes," https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/, (Accessed on 04/02/2022).

[32] A. Kratsios, "The universal approximation property," *Annals of Mathematics and Artificial Intelligence*, pp. 1–35, 2021.

[33] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[34] D. Bertsimas, D. B. Brown, and C. Caramanis, "Theory and applications of robust optimization," *SIAM review*, vol. 53, no. 3, pp. 464–501, 2011.

[35] "Software-defined networking and end-to-end visibility - epsglobal," https://www.epsglobal.com/business-solutions/sdn-and-network-monitoring, (Accessed on 09/20/2021).

[36] Z. Xiang, S. Pandi, J. Cabrera, F. Granelli, P. Seeling, and F. H. Fitzek, "An open source testbed for virtualized communication networks," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 77–83, 2021.

[37] "Managing resources for containers — kubernetes," https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/, (Accessed on 09/17/2021).

[38] S. Boyd, L. Xiao, A. Mutapcic, and J. Mattingley, "Notes on decomposition methods," *Notes for EE364B, Stanford University*, vol. 635, pp. 1–36, 2007.

[39] A. Nedić and A. Ozdaglar, "Subgradient methods for saddle-point problems," *Journal of optimization theory and applications*, vol. 142, no. 1, pp. 205–228, 2009.

[40] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[41] "Qos: Diffserv for quality of service overview configuration guide, cisco ios release 15m&t - overview of diffserv for quality of service [support]," Sep 2017. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/qos_dfsrv/configuration/15-mt/qos-dfsrv-15-mt-book/qos-dfsrv.html

[42] "Chapter 5: Advanced capabilities — 5g mobile networks: A systems approach version 1.1-dev documentation," https://5g.systemsapproach.org/disaggregate.html, (Accessed on 04/02/2022).