Data-driven Dynamic Optimization of Traffic Workload and Network Slicing for 5G Networks and beyond

El Hocine Bouzidi

Université Gustave Eiffel

Ecole Doctorale MSTIC

Laboratoire d'Informatique Gaspard-Monge LIGM

THÈSE présentée par :

EL Hocine BOUZIDI

Soutenue le : **5 Novembre 2021**

pour obtenir le grade de : **Docteur de L'Université Gustave Eiffel**

Discipline: **Informatique**

Optimisation Dynamique axée sur les données de la Charge et des Tranches Réseaux dans les Réseaux 5G et au-delà

JURY :

Thi Mai Trang Nguyen	Maître de conférences HDR, Sorbonne Université, France, Rapporteur
Yacine Ghamri-Doudane	Professeur, L3i / La Rochelle Université, France, Rapporteur
Stefano Secci	Professeur, Cnam Paris, France, Examinateur
Mohamed Faten Zhani	Maitre de conférences, ETS-Montréal, Université de Québec, Canada, Examinateur
Rami Langar	Professeur, Université Gustave Eiffel, France, Directeur de Thèse
Abdelkader Outtagarts	Chercheur, Nokia Bell Labs, France, Co-encadrant

University Gustave Eiffel

Doctoral School MSTIC

Gaspard-Monge Computer Laboratory LIGM

THESIS Presented by :

EL Hocine BOUZIDI

Defended on : **November 5, 2021**

to obtain the title of : **Doctor of Philosophy of the University Gustave Eiffel**

In : **Computer Science**

Data-driven Dynamic Optimization of Traffic Workload and Network Slicing for 5G Networks and beyond

Committee:

Thi Mai Trang Nguyen	Associate Professor, Sorbonne University, France, Reviewer
Yacine Ghamri-Doudane	Full Professor, La Rochelle University, France, Reviewer
Stefano Secci	Full Professor, Cnam Paris, France, Examinator
Mohamed Faten Zhani	Associate Professor, ETS-Montréal, Québec University, Canada, Examinator
Rami Langar	Full Professor, University Gustave Eiffel, France, Advisor
Abdelkader Outtagarts	Doctor, Nokia Bell Labs, France, Advisor

I would like to dedicate this thesis to my loving parents ...

Declaration

This is to certify that:

- the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university.
- this thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.
- all main sources of help have been acknowledged.
- the source is always given when quoting from the work of others.

EL Hocine BOUZIDI
July 2021

Acknowledgements

I would like to thank everybody who has made possible this Ph.D. thesis. First of all, I would like to thank my industrial supervisor, Dr. Abdelkader Outtargarts and my thesis director Professor Rami LANGAR for giving me the opportunity to pursue my studies and researches in Nokia Bell Labs and University Gustave Eiffel. Their patience and continuing guidance, support and encouragement helped me all the time of research and writing of this thesis.

I would also like to acknowledge all members of the CNTP department and the ones we have shared the space in our research laboratory, for their friendship and supports.

I would like to acknowledge the FUI SCORPION project (Grant no. 17/00464), the Université Gustave Eiffel and ANRT for providing funding and facilities to pursue my research.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Abstract

The Fifth Generation of mobile communications (5G) networks are expected to meet the ever increasing number of demands, connected devices and data traffic volume, which pushes network operators to their limits in terms of flexibility, elasticity and profitability. Software-Defined Networking (SDN), Network Function Virtualization (NFV), Network Slicing, Machine Learning (ML) and Cloud computing are considered as the key technologies to tackle these problems. However, the design and the maturity of these technologies raise new problems in terms of traffic engineering and routing optimization, network infrastructure sharing and distributed SDN control management. In this context, we address, in this thesis, the following problems: 1) how to reactively and proactively define the flow rules and where to place them in order to avoid network congestion, 2) how to create an isolated and efficient End-to-End (E2E) network slice by taking into account NFV and SDN to optimize the traffic routing and avoid congestion in a specific network slice, 3) how many SDN controllers needed, where to place them and their corresponding data plane domains in order to improve the performances, and 4) how to leverage intelligence via ML in SDN in order to solve complex problems arising in routing optimization, Network Slicing and controller placement, while guaranteeing the requested Quality of Service (QoS).

Our first contribution presents an SDN-based rules placement approach that aims to dynamically: i) predict traffic congestion by using mainly Neural Network (NN), then ii) learn optimal paths and reroute traffic to improve network link utilization by deploying a Deep Q-Network (DQN) agent. To do so, we first formulate the QoS-aware routing problem as an optimization problem whose objective is to minimize the total network delay and link utilization. Then, we propose, a simple yet efficient heuristic algorithm to solve it. Numerical results through emulation demonstrate that the proposed approach can significantly improve the performances in terms of decreasing the link utilization, the packet loss and the total network delay.

As a second contribution of this thesis, we design and implement an SDN based architecture for E2E network slicing, which proactively and dynamically adapts radio slices to the transport network slices. The developed architecture enables the creation, modification and continuity of radio and transport network slices, while considering their resource and QoS requirements. It leverages ML for predicting radio slices capacities, improving network resource utilization, and predicting congestion within each network slice. We also formulate this network slicing problem as an optimization problem, whose objective is to minimize the total network delay. Finally, we propose an efficient heuristic algorithm with low time complexity and high estimation accuracy to solve large problem instances. Experimental results using the OpenAirInterface (OAI) platform, Flexible and Programmable Platform for Software Defined Radio Access Network (SD-RAN) (FlexRAN), Open Network Operating System (ONOS) SDN Controllers and OVS demonstrate the efficiency of our approach in terms of guaranteeing low latency and high network throughput.

In the third and last contribution, we present a new method that dynamically computes the optimal number of controllers, determines their optimal locations, and at the same time partitions the set of data plane switches into clusters and assigns them to these controllers. First, we mathematically formulate the controller placement as an optimization problem, whose objectives are to minimize the controller response time, that is the delay between the SDN controller and assigned switches, the control load as well as the intra-cluster delay. Second, we propose a simple yet computationally efficient heuristic, called Deep Q-Network based Dynamic Clustering and Placement (DDCP), that leverages the potential of reinforcement and deep learning techniques to solve the aforementioned optimization problem. Experimental results using ONOS controller show that the proposed approach can significantly improve the network performances in terms of response time and resource utilization.

Keywords

5G, Software-Defined Networking (SDN), Network Function Virtualization (NFV), Network Slicing, Quality of Service (QoS), Open Networking Operating System (ONOS), Machine Learning (ML), Prediction, Neural Networks (NN), Deep Q-Network (DQN), Long Short-Term Memory (LSTM).

Table of contents

List of figures

List of tables

List of acronyms

1G First Generation . . . 13
2G Second Generation . . . 13
3D Three Dimensions . . . 40
3G Third Generation . . . 13
3GPP Third Generation Partnership Project . . . 13
4G Fourth Generation of mobile communications . . . 13
3D Three Dimensions . . . 40
5G Fifth Generation of mobile communications . . . iv
3D Three Dimensions . . . 40
AI Artificial Intelligence . . . 47
API Application Program Interface . . . 2
ARIMA Auto Regressive Integrated Moving Average . . . x
ADF Augmented Dickey Fuller . . . 31
APN Access Point Name . . . 16
ACF Autocorrelation Function . . . 52
AR Auto-Regressive . . . 31
ARMA Auto-Regressive Moving Average . . . 31
ARAR Auto-regressive Auto-regressive . . . 38
BBU BaseBand Unit . . . 16
B5G Beyond 5G networks . . . 91
CC Cloud Computing . . . 13
CN Core Network . . . 15
CPU Central Processing Unit . . . 91
C-RAN Cloud Radio Access Network . . . 16

Résumé de la thèse

1 Introduction

Dans cette thèse, nous nous sommes intéressés à l 'ingénierie du trafic réseau et, à d'optimisation du routage, du tranchage réseau et de la gestion distribuée des contrôleurs SDN. Dans ce chapitre, nous résumons le contexte et les contributions de cette thèse. Nous commencerons par le contexte et les motivations de nos travaux. Puis, nous décrirons les approches qu'on a proposées.

2 Contexte et motivations

Avec le développement de nouvelles technologies telles que la 5G, le trafic réseau devrait croître à un rythme exponentiel, en raison de la large gamme d'applications, avec des exigences strictes et hétérogènes, telles que des applications de streaming en temps réel, des communications massives, des communications haut débit et à faible latence, des applications big data et cloud, des services Over The Top (OTT). De plus, avec l'avènement de nouvelles technologies telles que NFV [59], les flux de trafic dans les réseaux de paquets peuvent avoir besoin de passer par différents équipement intermédiaires matérielles et logicielles telles que Network Address Translators (NAT), proxy, pare-feu. Cela rend le Traffic Engineering (TE) difficile, en particulier le routage compatible QoS et la télémétrie réseau où les flux de trafic doivent être attribués à chaque réseau en fonction de Service Level Agreement (SLA) et les exigences de flux doivent être satisfaites en termes de collecte des statistiques avec précision depuis le trafic transitant par les réseaux de paquets.

De plus, ces applications necessitent une infrastructure programmable et flexible, qui permet de controler et d'orchestrer les ressources. Elle permet également le tranchage du réseau physique [19], qui qui prend en charge l'optimisation des services heterogenes. Ces services sont exécuté dans des réseaux virtuels (Virtual Network ou Virtual Network (VN)) independant. En particulier, dans les réseaux mobiles, le tranchage du réseau est généralement applicable à la fois dans les domaines Radio Access Network (RAN) [1] et dans le cœur du réseau. Les tranches radio partagent des ressources radio entre divers services (c'est-à-dire IoT, enhanced Mobile Broadband (eMBB)) en réservant une partie appropriée de la bande passante (certain nombre de RB) à utiliser à partir de cette tranche pour un intervalle de temps précis. D'autre part, le tranchage du réseau de transport consiste à créer des Packet Networks (PN) isolés composés de commutateurs physiques ou virtuels attribués à un

tenant par le fournisseur de réseau qui, à son tour, possède les ressources physiques correspondantes. Cependant, la création d'une tranche de réseau de bout en bout isolée et efficace reste difficile, en particulier lorsque l'on considère la qualité de service QoS au niveau du routage et la télémétrie réseau.

SDN [116] est l'une des technologies émergentes clés pour la vision 5G, permettant aux fournisseurs de contrôler et d'orchestrer leurs ressources via le tranchage du réseau tout en considérant à la fois le routage compatible à la QoS et à la télémétrie réseau. SDN permet la dynamicité, l'automatisation, la flexibilité et la gestion centralisée du réseau sous-jacent contrairement aux réseaux traditionnels, en séparant le plan de contrôle qui est chargé de prendre les décisions de routage et le plan de transfert. Cette communication est activée via Application Program Interface (API) tel que API sur l'interface nord et le protocole OpenFlow sur l'interface sud. Bien que la séparation du contrôle et des plans de données apporte des avantages techniques, elle peut entraîner plusieurs défis tels que le nombre de contrôleurs nécessaires, leurs problèmes de placement correspondants, quels dispositifs de plan de données doivent être contrôlés et par quel contrôleur.

Bien que SDN apporte des avantages permettant la mise en place du tranchage réseau tout en considérant à la fois le routage compatible à la QoS et à la télémétrie réseau de manière distribuée, il a toujours besoin de connaissances pour la gestion du réseau. En fait, les éléments actuels du plan de données sont équipés de capacités de stockage et de calcul améliorées qui permettent une surveillance efficace en temps réel (c'est-à-dire la télémétrie réseau), en fournissant au plan de contrôle des données de configuration et d'état du réseau ainsi que des paquets et des flux en temps réel. informations des granularités. De cette manière, l'intégration de l'intelligence via l'apprentissage automatique ML[167] au plan de contrôle SDN est cruciale pour résoudre les problèmes complexes qui se posent dans le tranchage réseau, et l'optimisation du routage tout en garantissant la QoS demandée. Cependant, en raison du nombre élevé de flux de trafic provenant de la large gamme d'applications, la taille de l'espace de solution ainsi que la complexité du problème d'optimisation sont augmentées puisque le plan de contrôle est responsable de la gestion du réseau sous-jacent et de la résolution du problème. Certains auteurs [45] ont introduit le Plan de Connaissance Knowledge plane (KP) au paradigme conventionnel SDN, tout en effectuant la telemetrie reseau sur une plate-forme Network Analytics (Network Analytics (NA)) centralisée. De cette façon, le plan de connaissance Knowledge plane (KP) exploite les données provenant du NA comme entrée pour alimenter les algorithmes ML, tels que les techniques Deep Reinforcement Learning (DRL), qui les convertiront en forme de connaissance.

Dans ce contexte, cette thèse aborde le problème de la construction et de l'exécution d'un framework pour assurer le contrôle prédictif des ressources supportant plusieurs tranches en utilisant des contrôleurs distribués et coopératifs SDN suivant le paradigme KDN. Tout d'abord, nous nous concentrons sur TE dans SDN, plus précisément, le routage compatible QoS et la télémétrie réseau. Ensuite, sur la base de la solution proposée, nous abordons le problème du tranchage de réseau de bout en bout dans les réseaux mobiles, plus précisément, nous adoptons SDN et KDN comme technologies habilitantes clés pour concevoir et optimiser la prochaine génération réseaux mobiles 5G. Enfin, nous nous concentrons sur l'aspect distributif du plan de contrôle dans les réseaux SDN, à cette fin, nous

abordons le problème du clustering et du placement des éléments des plans de contrôle et de données SDN.

Dans ce qui suit, nous résumons les différentes contributions dans la Section 3. Ensuite, nous présentons l'organisation de cette thèse dans la Section 4.

3 Contributions

Nous avons abordé dans cette thèse, les problèmes suivants : 1) comment définir de manière réactive et proactive les règles de flux et où les placer afin de supprimer la congestion du réseau, 2) comment créer des tranches réseaux de bout en bout isolés et efficaces en prenant en compte NFV et SDN pour optimiser le routage du trafic et éviter la congestion dans une tranche de réseau spécifique, 3) combien de contrôleurs SDN nécessaires, où les placer et leurs domaines de plan de données correspondants afin d'améliorer les performances, et 4) comment tirer parti de l'intelligence via ML dans SDN afin de résoudre des problèmes complexes survenant dans l'optimisation du routage, le découpage réseau et le placement des contrôleurs, tout en garantissant la qualité de service QoS demandée. Ces problèmes sont traités en trois contributions comme suit:

3.1 Optimisation du routage réseau basée sur l'apprentissage profond par renforcement (Deep Q-Network) et la prédiction du trafic

Comme première contribution, nous proposons une solution pour ingénierie du trafic réseau appelé : Optimisation du routage réseau basée sur l'apprentissage profond par renforcement (Deep Q-Network) et la prédiction du trafic DTPRO, dans laquel nous optimisons le routage du flux du réseau via DQN et la prévision du trafic. Plus précisément, notre solution proposée se compose de trois phases principales. Premièrement, nous optimisons dynamiquement le routage des flux dans le réseau en formant un agent DQN. Deuxièmement, nous prédisons la congestion et ajustons la fonction de récompense DQN pour fournir de meilleures configurations de routage. Enfin, nous acheminons le trafic réseau sur la base d'un ensemble de poids de liens trouvés par l'agent DQN, et en même temps réacheminons ceux existants loin des chemins encombrés en résolvant un problème d'optimisation. Le problème est formulé comme un Programme Linéaire LP, et représente le problème de placement des règles de flux, où l'objectif est de minimiser le retard total du réseau, la perte de paquets et l'utilisation de la liaison. De plus, nous proposons une heuristique qui interagit avec l'agent DQN et le module du prédiction de trafic afin de résoudre le problème formulé et d'optimiser les performances du réseau. Les résultats expérimentaux montrent que l'approche proposée offre une amélioration prometteuse par rapport aux algorithmes de routage statique traditionnels.

Cette contribution fait l'objet de deux publications de conférence dans la Global Communications Conference (GLOBECOM) [34], [35] et une éventuelle publication de revue dans Elsevier Journal of Network and Computer Applications (JNCA) [33].

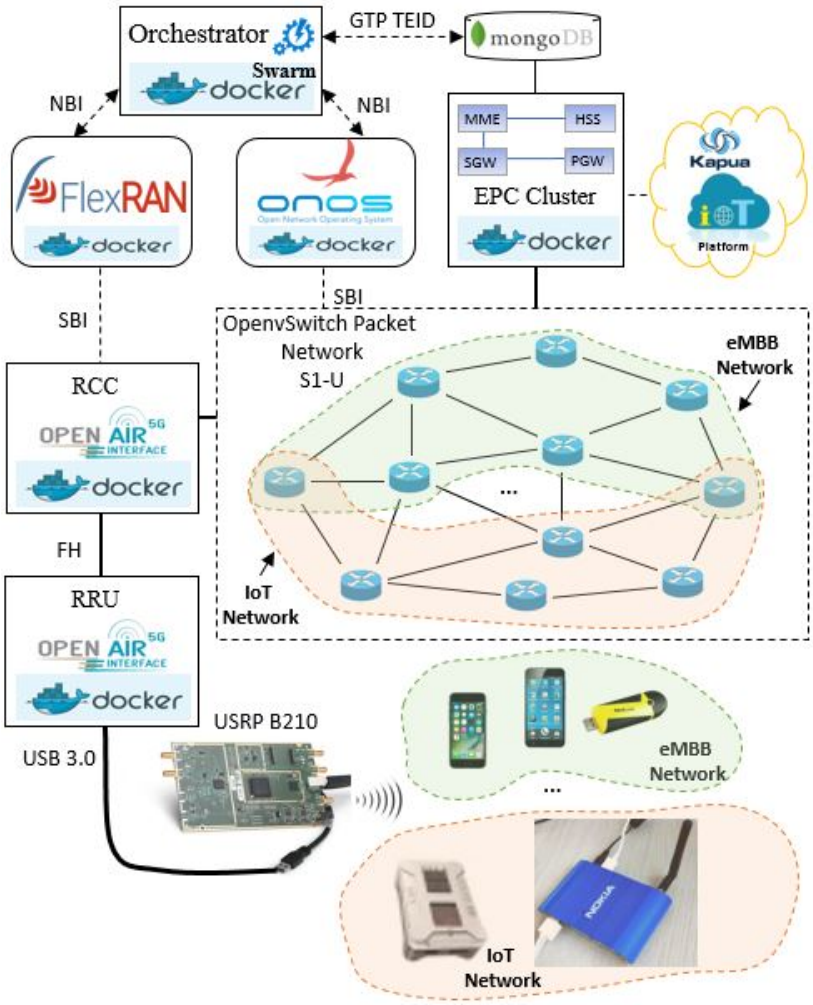

Fig. 1 Framework du tranchage de bout en bout

3.2 Apprentissage en ligne pour le tranchage prédictif de bout en bout du réseau dans les réseaux 5G

Dans la deuxième contribution, nous abordons la conception et la mise en œuvre d'une nouvelle architecture basée sur les techniques SDN et ML pour permettre la création, la modification et la continuité de tranches de réseaux radio et de transport, tout en considérant leurs performances et exigences en termes de QoS, comme le montre la figure 1. Pour ce faire, notre solution s'appuie sur l'outil OAI, FlexRAN, ONOS et une version étendue de Open Virtual Switch (OVS) que nous avons modifié pour lire les paquets GPRS Tunnelling Protocol (GTP). Deuxièmement, la solution proposée permet un partage efficace des ressources RAN et du réseau de paquets en ajustant de manière proactive les capacités des tranches radio et en les adaptant aux tranches de réseau de paquets correspondantes, puis un équilibre de la charge de trafic et de la prévention de la congestion a été proposé pour améliorer le routage et éviter la congestion au sein de chaque tranche de réseau de

paquets. À cette fin, nous formulons mathématiquement le problème comme un Linear Program (LP) qui vise à minimiser la latence totale du réseau et proposons un algorithme de placement de règle de flux et d'équilibrage de charge et de prédiction de congestion (CLR) pour le résoudre avec une faible complexité temporelle et haute précision d'estimation. Les résultats expérimentaux montrent la faisabilité de la solution proposée de gestion de la continuité de découpage E2E en temps réel. Plus précisément, les résultats montrent la surperformance de l'utilisation des techniques ML en temps réel sur d'autres schémas en termes d'augmentation du débit et de diminution de la perte de paquets et de la latence. Cette contribution fait l'objet d'une publication de conférence dans la Conférence Internationale des Communications (ICC) [36].

3.3 Partitionnement dynamique de commutateurs du réseau définis par logiciel et placement de contrôleurs à l'aide de l'apprentissage par renforcement profond

Comme troisième contribution, nous proposons une approche pour déterminer, dans un réseau SDN, le nombre optimal de contrôleurs, leurs emplacements optimaux et le partitionnement optimal des commutateurs de plan de données tout en considérant le délai de propagation entre les commutateurs et le contrôleur, l'utilisation des ressources du contrôleur, le délai intra-cluster. À cette fin, nous formulons d'abord mathématiquement le problème de placement, où le problème d'optimisation correspondant est de minimiser le délai entre les contrôleurs et les commutateurs, l'utilisation des ressources de contrôle et le délai intra-cluster. Comme le problème d'optimisation formulé est un problème Non-deterministic Polynomial-time Hard (NP-Hard), dont la solution optimale est très difficile à obtenir en général. Pour résoudre le problème formulé, nous proposons d'abord de modéliser le problème comme un Markov Decision Process (MDP) et de le résoudre par une approche d'apprentissage par renforcement profond. Ensuite, nous proposons une heuristique efficace appelée Partitionnement dynamique de commutateurs du réseau définis par logiciel et placement de contrôleurs à l'aide de l'apprentissage par renforcement profond (DDCP) qui interagit dynamiquement avec l'agent d'apprentissage par renforcement profond pour obtenir le partitionnement optimal et le placement des contrôleurs. Les résultats de la simulation montrent l'efficacité de l'utilisation du DQN pour le regroupement et le placement des contrôleurs et des commutateurs. De plus, le DQN surpasse la méthode de partitionnement bien connue K-means, en termes de diminution du délai de contrôle, de la charge de contrôle et de la latence intra-cluster. Notez que cette contribution a été soumise pour publication dans Computer Networks [32].

4 Organisation de la thèse

Cette thèse est organisée en 7 chapitres. Les chapitres de base sont dérivés de différentes publications de recherche publiées au cours des travaux de doctorat comme suit:

Chapitre 2 fournit un aperçu de l'architecture, des défis et des concepts de 5G, en mettant davantage l'accent sur le GTP pour le tranchage réseau. Ensuite, il donne un aperçu de l'architecture SDN et du protocole OpenFlow. Enfin, il met en évidence le paradigme KDN en mettant davantage l'accent sur les méthodes ML utilisées pour la prédiction et l'optimisation du trafic réseau.

Chapitre 3 présente l'état de l'art, dans lequel nous soulignons et revoyons la problématique de la thèse, identifions les lacunes et proposons nos contributions.

Chapitre 4 étudie le problème de routage basé sur le paradigme KDN. Plus précisément, nous présentons notre première contribution sur l'ptimisation du routage réseau basée sur l'apprentissage profond par renforcement (Deep Q-Network) et la prédiction du trafic (DTPRO).

Chapitre 5 aborde le problème du tranchage de bout en bout du réseau basé sur les méthodes proposées dans le chapitre 4. Plus précisément, nous présentons notre deuxième contribution sur l'apprentissage en ligne pour le tranchage prédictif de bout en bout du réseau dans les réseaux 5G.

Chapitre 6 étudie le problème de la détermination du nombre de contrôleurs, leur placement ainsi que du partitionnement des commutateurs dans le plan de données dans une architecture SDN distribuée, en présentant notre dernière contribution sur le partitionnement dynamique de commutateurs du réseau définis par logiciel et placement de contrôleurs à l'aide de l'apprentissage par renforcement profond.

Chapitre 7 conclut cette thèse et synthétise les objectifs de la thèse et les contributions associées et présente une perspective pour des travaux futurs.

Chapter 1

Introduction

Contents

1.1 Context and Motivation

WITH the development of new technologies such as 5G, the network traffic is expected to grow at an exponential rate, due to the wide range of applications with stringent and heterogeneous requirements, such as real-time streaming applications, massive machine-type communications, ultra-reliable low-latency, enhanced mobile broadband communications, big data and cloud applications, OTT services. In addition, with the advent of new technologies like NFV [59], the traffic flows in packet networks may need to pass through different hardware and software applications middle-boxes such as NAT, proxies, firewalls. This makes the TE challenging, in particular, the QoS-aware routing and Network Telemetry where the traffic flows should be assigned to each network based on SLA and the flows requirements should be satisfied by accurately and timely measuring the traffic volumes flowing through the packet networks.

Furthermore, these applications require a programmable and a flexible infrastructure, which allows to control and orchestrate the resources and enables the sharing of the same physical network infrastructure through the concept of Network slicing [19], which enables optimal support for heterogeneous services, by running them in an independent VN, on a common shared Physical Network Infrastructure (PNI), such that each VN is allocated a fixed or dynamic portion of the PNI resources. In particular, in mobile networks, Network slicing is generally applicable in both RAN [1] referred to as radio slicing and transport and Core Network domains, referred to as transport network slicing. Radio slices share radio resources among diverse services (i.e., IoT, eMBB) by reserving an appropriate portion of bandwidth (i.e., a number of RB) to be used from that slice for a specific time interval. On the other hand, transport network slicing consists in creating isolated PN composed of physical or virtual switches assigned to a tenant by the network provider that, in turn, owns the corresponding physical resources. However, creating an isolated and efficient end-to-end network slice remains challenging, particularly when considering the QoS-aware routing and Network Telemetry.

SDN [116] is one of the key emerging technologies for the 5G vision, allowing the providers to control and orchestrate their resources through Network Slicing while considering both the QoS-aware routing and Network Telemetry. SDN allows dynamicity, automation, flexibility and centralized management of the underlying network contrary to traditional networks, by separating the control plane that is responsible for making routing decisions and the forwarding plane, that is only required to perform packet forwarding according to the received routing strategies from the control plane. This communication is enabled via API such as API on the northbound interface and the OpenFlow protocol on the southbound interface. Even the separation of the control and the data planes brings technical benefits, it can cause several drawbacks such as the number of controllers needed, their corresponding placement problems and which data plane devices to be controlled by which controller.

Even SDN paves the way to efficiently enable Network Slicing while considering both QoS-aware routing and Network Telemetry in a distributed fashion, it still needs the knowledge for the network management. In fact, current data plane elements are equipped with improved storage and

computing capabilities which enable efficient real time monitoring (i.e., Network Telemetry), by providing the control plane, with configuration and network state data as well as real-time packet and flow-granularity information. In this way, incorporating intelligence via ML [167] to the SDN control plane is crucial to solve complex problems arising in Network Slicing, routing optimization while guaranteeing the requested QoS. However, due to the high number of traffic flows flowed from the wide range of applications, the size of the solution space as well as the complexity of the optimization problem are increased since the control plane is responsible of managing the underlying network and solving the problem with a global view. KDN [45], is well placed to cope with these challenges, by introducing the Knowledge plane KP to the conventional SDN paradigm, while reporting the Network Telemetry to a centralized NA platform. In this way, the KP plane exploits the data coming from the NA as input to be fed to ML algorithms, such as DRL techniques, which will convert them to the form of knowledge.

In this context, this thesis addresses the problem of building and running a framework to ensure the predictive control of resources supporting multiple slices using distributed and cooperative SDN controllers following the KDN paradigm. First, we focus on TE in SDN, more specifically, QoS-aware routing and Network telemetry. Then, based on the proposed solution, we address the problem of E2E Network Slicing in mobile networks, more specifically, we adopt SDN and KDN as key enabling technologies to design and optimize next generation mobile networks 5G. Finally, we focus on the distributive aspect of the control plane in SDN networks, to this end, we address the problem of clustering and placement of SDN control and data planes elements. In what follows, we summarize the different contributions in Section 1.2. Then, we present the organization of this thesis in Section 1.3.

1.2 Contributions

This thesis contributions can be broadly classified into 3 major categories. The first contribution addresses the QoS-aware routing in Software Defined Networking. The second contribution extends the first one to be used in an E2E Network Slicing approach. The last contribution addresses the optimal controllers and switches placement in a distributed SDN architecture. In what follows, we present in a nutshell these contributions.

1.2.1 Deep Q-Network and Traffic Prediction based Routing Optimization in Software Defined Networking

As a first contribution, we propose a dynamic and efficient TE scheme, called DTPRO for Deep Q-Network (DQN) and Traffic Prediction based Routing Optimization, in which we optimize the network flow routing based on DQN and traffic prediction. Specifically, our proposed solution consists of three main phases. Firstly, we dynamically optimize the flow routing in the network by training a DQN agent. Secondly, we predict congestion and adjust the DQN reward function to provide better routing configurations. Finally, we route the network traffic based on a set of link weights found by the

DQN agent, and at the same time reroute the existing ones away from the congested paths by resolving an optimization problem. The problem is formulated as a LP, the formulated problem represents the flow rules placement problem, where the objective is to minimize the total network delay, packet loss and link utilization. Moreover, we propose an heuristic that interacts with the DQN agent and the traffic prediction in order to solve the formulated problem and optimize network performances. Experimental results show that the proposed approach provides a promising enhancement against traditional static routing algorithms. This contribution is the object of two conference publications in the Global Communications Conference (GLOBECOM) [34], [35] and a possible journal publication in Elsevier Journal of Network and Computer Applications (JNCA) [33].

1.2.2 Online based learning for predictive end-to-end network slicing in 5G networks

As a second contribution, we address the design and implementation of a novel architecture based on SDN and ML techniques for enabling creation, modification and continuity of radio and transport network slices, while considering their performances and QoS requirements. To do so, our solution relies on OAI tool, FlexRAN, ONOS and an extended version of OVS that we patched to handle GTP packets. Second, the proposed solution enables efficient sharing of RAN and Packet Network resources by proactively adjusting radio slices capacities and adapt them to the corresponding Packet Network slices, then a balancing of traffic load and congestion prevention have been proposed for improving routing and avoiding congestion within each Packet Network slice. To this end, we mathematically formulate the problem as a LP that aims to minimize the total network delay and propose a Congestion prediction and Load balancing Rule (CLR) placement algorithm to solve it with low time complexity and high estimation accuracy. Experimental results show the feasibility of the proposed solution of handling E2E slicing continuity in real-time. Specifically, results show the outperformance of using ML techniques in real-time on other schemes in terms of increasing throughput and decreasing packet loss and latency. This contribution is the object of a conference publication in the International Conference on Communications (ICC) [36].

1.2.3 Dynamic Clustering of Software Defined Network Switches and Controller placement using Deep Reinforcement Learning

As a third contribution, we propose an approach to determine, in an SDN network, the optimal number of controllers, their optimal placements and the optimal clustering of data plane switches while considering the propagation delay between the switches and the controller, the controller resource utilization, the intra-cluster delay and the dynamicity of the network. To this end, we first mathematically formulate the placement problem, where the corresponding optimization problem is to minimize the delay between controllers and switches, the control resource utilization and the intra-cluster delay. As the formulated optimization problem is an NP-Hard problem, the optimal solution of which is very difficult to obtain in general. To solve the formulated problem, we first propose to model the problem as a MDP and solve it by a deep reinforcement learning approach. Then,

we propose an efficient heuristic called Deep Q-Network based Dynamic Clustering and Placement (DDCP) that dynamically interacts with the Deep Reinforcement Learning agent to get the optimal clustering and controllers placement. Simulation results, show the efficiency of using the DQN for controllers and switches clustering and placement. Moreover, the DQN outperforms the well known clustering method K-means, in terms of decreasing the control delay, the control load and the intra-cluster latency. Note that this contribution has been submitted for publication in Computer Networks [32].

1.3 Outline

This thesis is organized in 7 chapters, the core chapters are derived from different research publications issued during the PhD works as follows:

Chapter 2 first, provides an overview of the 5G architecture, challenges and concepts, with more focus on the GTP for Network Slicing. Then, it provides an overview of the SDN architecture and OpenFlow protocol. Finally, it highlights the KDN paradigm with more focus on ML methods used for network traffic prediction and optimization.

Chapter 3 presents the state of the art, in which we highlight and review the problematic of our thesis, identify the shortcoming and propose our contributions.

Chapter 4 investigates the QoS-aware routing problem based on the KDN paradigm. More specifically, we present our first contribution, namely, Deep Q-Network and Traffic Prediction based Routing Optimization in Software Defined Networking (DTPRO).

Chapter 5 addresses the problem of E2E Network Slicing based on the methods proposed in Chapter 4. More specifically, we present our second contribution, namely, Online based learning for predictive E2E network slicing in 5G networks.

Chapter 6 investigates the problem of determining the number of controllers, their placement as well as the data plane devices clustering in a distributed SDN architecture, by presenting our last contribution, namely, Dynamic Clustering of Software Defined Network Switches and Controller placement using Deep Reinforcement Learning.

Chapter 7 concludes this thesis, synthesizes the thesis objectives and the associated contributions and presents a perspective for future works.

Chapter 2

5G Challenges and Technologies Enablers Background

Contents

2.1 Introduction

THE fifth generation networks 5G are expected to meet the rising and future data traffic demand while accommodating a multitude of services, with stringent and heterogeneous requirements such as latency, security, data rates, energy consumption, reliability, etc. Mobile networks in particular, are envisioned to accommodate new services such as Virtual Realities (VR/AR), which have various requirements. In this way, SDN, NFV, Network Slicing, Cloud Computing (CC) and ML are considered as the most important key technologies to meet these requirements and achieve the goals of 5G.

In this context, this chapter provides an overview of the main technologies used to optimize network performances. We first start by an overview of the 5G requirements and challenges in Section 2.2 and its detailed architecture in Section 2.3, then, we overview the SDN technology in Section 2.4, followed by an overview of a specific set of ML techniques in Section 2.5. Finally, Section 2.6 concludes this chapter.

2.2 5G Requirements and Challenges

The mobile wireless evolution progressed through several generations. The First Generation (1G) in 1981 and the Second Generation (2G) with analog voice communication, then, there was the Third Generation (3G) with multimedia support, spread spectrum transmission. After that, the Fourth Generation of mobile communications (4G) with all Internet Protocol (IP) Switched networks in 2011. The 4G, which refers to Long Term Evolution (LTE), firstly introduced by the Third Generation Partnership Project (3GPP) release 8, in the EPS project [139]. Its objective is to enable a flat IP-based network as well as inter-operability between different sorts of networks by standardizing network elements and interfaces. The next generation mobile networks referred to as the 5G.

The 5G networks are expected to meet a wide range of use-cases scenarios with heterogeneous QoS requirements, which, according ITU-R [138], are classified as: i) eMBB which provides a huge bandwidth for mobile devices on demand, ii) massive Machine Type communications (mMTC) to connect a very large number of IoT devices, and iii) ultra Reliable Low Latency communications (uRLLC) which supports low-latency transmissions with high reliability.

Therefore, these requirements bring new challenges due to the additional management and control complexity increases in SDN and NFV architectures, the share of computing resources among multiple tenants and the need for automation of network operations to reduce the manpower load. In what follows we describe some challenges.

- **Heterogeneous Demands**: the functionalities and architecture of future 5G networks are expected to be agile in order to provide services per requirement and accommodate the use cases, in terms of throughput and mobility in eMBB, transmission latency and reliability in uRLLC, and energy efficiency in mMTC.

- **Network Slicing and Agility**: Network slicing aims to assure service customization, isolation and multi-tenancy support on a common physical network infrastructure by enabling logical and physical separation of network resources [16], encouraging thus vertical players to use operators' networks rather than deploying their own infrastructure. On the other hand, the network agility aims to assure the flexibility to rapidly change infrastructure's capability.
- **Big data**: the ever growing heterogeneous traffic, mobile network subscribers and online services have lead to a huge flood of data which can be defined as volume, variety and velocity [131].
- **Cloudification and resource orchestration**: which can be achieved by enabling deployments on private, public or hybrid cloud as well as adopting service orchestration and processes automation.

Figure 2.1 shows the 5G ecosystem, which is characterized by a set of use cases in the top of the triangle. To achieve these use cases, a multitude of services, with stringent and heterogeneous requirements should be accommodated. In this way, a set of enablers should be considered to realize 5G concepts such as Network Slicing. In the following sections, we detail some concepts and enablers used in our work.

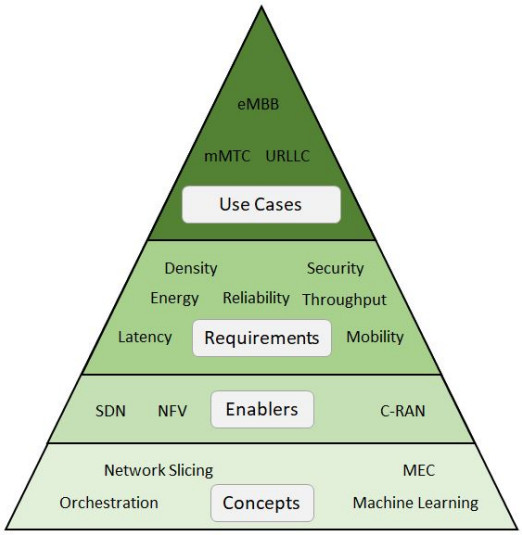

Fig. 2.1 5G Ecosystem (based on [98])

2.3 Detailed 5G E2E Architecture

3GPP has specified 8 possible configurations or Options for connecting to an Evolved Packet Core (EPC) or new 5G core network [84]. The architectural options identified encompass all potential deployment scenarios starting from the legacy LTE to full-fledged 5G. These options are grouped

into two categories, the first one called Non Stand-Alone (NSA) in which the options using dual Connectivity are grouped together where 5G radio access technology (New Radio (NR)) and LTE are used simultaneously to provide radio access, the second one called Standalone (SA) characterized by using only the new radio access technology. Moreover, the Option 3 allows the re-use of existing EPC Core functionality, which is expected to be initially used to deploy 5G. In this way, we extending an existing 4G LTE platform by opting for the Option 3 to deploy an E2E 5G architecture, as depicted in figure 2.2.

Figure 2.2 shows the detailed 5G E2E architecture [98], which consists of the RAN, and the Core Network (CN) EPC. By the mean of network elements and standardized interfaces between them in both the core and access network, it will be possible to provide the user with IP connectivity to a Packet Data Network (PDN) for running services such as IP Multimedia Subsystem (IMS) services, accessing the Internet as well as to allow multi-vendor interoperability. The core network as well as the access network parts are described bellow.

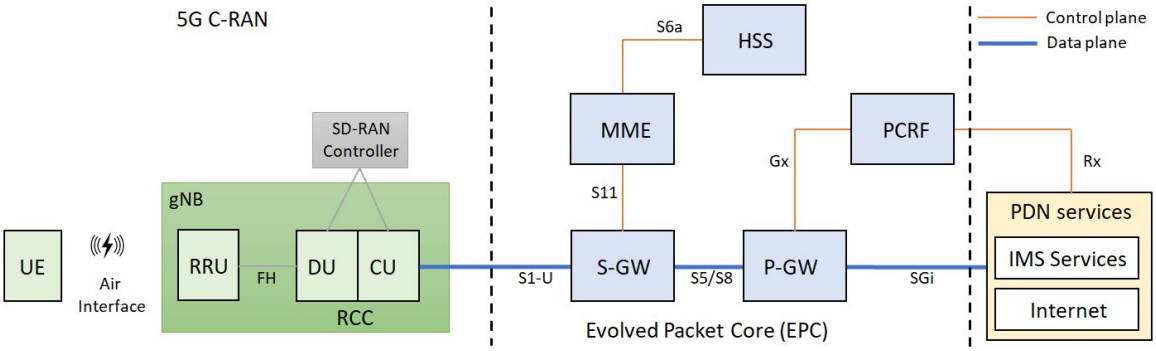

Fig. 2.2 Detailed 5G E2E Architecture (option 3) (based on [98])

2.3.1 The core network EPC

As shown in figure 2.2, the EPC represents the fundamental part of the entire system, since it lays between the access network and the external PDN services that a User User Equipment (UE) wants to communicate with. Its main responsibilities include the overall control of the UE and establishment of the bearers. The EPC consists of multiple nodes, where the main ones being Mobility Management Entity (MME), Home Subscriber Server (HSS), Serving Gateway (SGW) and Packet Data Network Gateway (PGW). Besides of these main nodes, the EPC also may include other elements such as the Policy and Charging Rules Function (PCRF). These nodes are described as follows [98]:

- **MME:** it is the control node located at the edge of the core network EPC, its main functions include: 1) processing the signaling between the UE and the core network using Non Access Stratum (NAS) protocol, 2) user authentication and roaming with HSS using *S6a* interface, 3) establishment, maintenance and release of the bearers, 4) establishment of the connection and

security between the network and UE, and 5) mobility management (tracking and the paging of UE).

- **HSS:** is a database that contains permanent subscription details, it provides user authentication information and profiles to the corresponding MME through the *S6a* interface. Furthermore, it helps service providers, managing in real time their subscribers, by holding information such as the PDN address (i.e., in the form of an Access Point Name (APN)), calls and IP session Setup and the identity of the MME corresponding to the current attached or registered subscribers.

- **SGW:** is the point of interconnection between the access network and the EPC using the *S1-U* interface, its main functions include: 1) dealing with the user plane, by transporting the incoming and outgoing IP data traffic between the UE and the external networks, 2) serving as a mobility anchor point for the UE when it moves from one Next Generation Node-B (gNB) to another, 3) buffering the data destined to the UE in the downlink, when a UE is in idle mode, and 4) serving as the mobility anchor point for inter-3GPP handover such as inter-working with General Packet Radio Service (GPRS).

- **PGW:** is the point of interconnection between the EPC and the external networks using the *SGi* interface. The main functions supported by PGW include: 1) IP address allocation for the UE, 2) enabling the forwarding of user data traffic to and from external PDN networks, 3) performing policy enforcement, packet filtering and charging according to the rules provided by the PCRF, and 4) serving as the mobility anchor point for handover between 3GPP and non-3GPP technologies.

- **PCRF:** is the policy management entity designed to specify the service policy and the QoS information, enabling thus more flexibility in engineering and policy control of resources and guaranteeing subscribers services according to their contracts. Its main functions include: 1) Enabling service-based policy control, 2) Providing charging information and rating parameters, and 3) Deciding what policies to install in the network based on subscription and network parameters.

2.3.2 The access network

The access network deployment refers to Cloud Radio Access Network (C-RAN)[41], in which, the traditional base station (BS) functionality is splitted into centralized BaseBand Unit (BBU) and the Remote Radio Head (RRH). The BBU can be implemented on commodity hardware and executed on a virtualized environment, that makes it possible to be pooled in a cloud data center. On the other hand, the RRHs are present at cell sites. Due to scalability and performance requirements, Next Generation Fronthaul Interface (NGFI) redefined the baseband processing split between BBU and RRH, where BBU is redefined as Radio Cloud Center (RCC), and RRH as Remote Radio Unit (RRU). Furthermore, to facilitate RAN virtualization and reduce fronthaul capacity and latency requirements,

the BBU is still splitted into Digital Unit (DU) and Central Unit (CU). Note that, in our work, we make use of the Co-located CU and DU forming together one block [11].

Moreover, the C-RAN is deployed following an SD-RAN architecture, in which the RAN control and data planes are separated using a Radio Controller. In this thesis, we make use of the SD-RAN controller FlexRAN [63] which is the first open-source flexible and programmable SD-RAN platform, which is implemented as an extension to a modified version of the OAI LTE platform [63]. It is designed for the mobile RAN and implemented in an SDN-based fashion by incorporating an API to separate control and data planes, where its objective is to make real-time RAN control applications feasible. In the context of RAN platforms. FlexRAN provides a great degree of flexibility to easily and dynamically coordinate among base stations, while adapts control over time and easier evolution to the 5G networks following SDN/NFV principles.

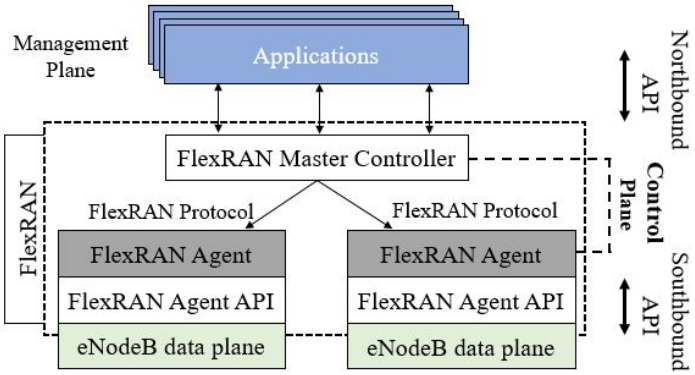

Fig. 2.3 SD-RAN FlexRAN Architecture (based on [63])

Figure 2.3 shows the overall architecture of the SD-RAN FlexRAN controller, which is made up of three main components: Control Plane, Data Plane and Management Plane. The Control Plane consists of a Master Controller connected with a set FlexRAN agents located in the gNBs through the FlexRAN Protocols. Based on these protocols, the agent sends the gNB statistics, configurations and events to the master and the latter issues control commands that define the operation of the agents. Moreover, as the master controller and the FlexRAN agents located in separated hosts, the communication channel would be a high-bandwidth and low-latency channel (e.g., optical fiber).

Similar to the SDN southbound interface (i.e., OpenFlow protocol), the control and data planes are separated based on the FlexRAN Agent API. On the hand, contrary to SDN controllers logic found in the wired domain, the data plane is charged to apply the Medium Access (MAC) scheduling decision dictated by the master controller, such as the fraction of RBs that the corresponding user connection type is allowed to use.

The management plane lies on top of the master controller and communicates with it through the northbound API, which incorporates the programmability aspect of the FlexRAN design. Its RAN control and management applications leverages the statistics and events gathered from the gNBs in

the FlexRAN control plane to modify the state of the network infrastructure (gNBs and UEs) such as monitoring applications, modifying the state of the RAN (e.g., MAC scheduler).

In this context, the access network consists of a set of gNB interconnected by the means of the $X2$ interface and connected to the SGW and MME, in the core network EPC, by the means of $S1$-U and $S1$-MME interfaces respectively, as shown in figure 2.2. The C-RAN is responsible of radio resource management, its main functions include: 1) radio admission control, 2) gNB measurement, configuration and provision, 3) scheduling and dynamic allocation of resources to UEs, 4) connection mobility control, 5) radio bearer control, and 6) inter cell interference coordination.

2.3.3 EPS Bearer

Figure 2.4 shows the EPS bearer service architecture [58]. The E2E IP connection between a UE and a PDN corresponds to EPS session, which is identified by the combination of PDN identifier (PDN ID or APN) and the corresponding UE IP address. The EPS session IP traffic can be classified into different classes and transported as an aggregated data flows through different logical associations between the UE and the PGW, which refers to as EPS Bearers. The EPS Bearer is a pipe that transports IP packets that have a common QoS, over the LTE network, which is allocated by the MME and corresponds to a specific QoS Class Identifier (QCI), where the QCI is used as a reference to a specific packet forwarding behavior (e.g. packet loss rate, packet delay budget). There are two types of bearers:

- **Default Bearer:** corresponds to the bearer established at the UE initial attache of the UE to the network and the connection to the corresponding PDN. A default Bearer provides the users with always-on IP connectivity that remains established throughout the lifetime of the EPS session or PDN connection.
- **Dedicated Bearer:** is established and associated with a default bearer of the same EPS connection and dedicated for applications based on QoS parameter. The user traffic flows are mapped to the corresponding Bearer by the means of the PCRF node.

As shown in figure 2.4, an EPS bearer is composed of three segments:

- **Data Radio Bearer (DRB):** established between the gNB and the UE over the radio interface and handled by the gNB such as association of DRB IDs to users.
- **S1 Bearer:** established between the gNB and the SGW over the $S1$-U interface and transports user data based on the GTP.
- **S5/S8 Bearer:** established between the SGW and the PGW over the $S5$/$S8$ interface and transports user data based on the GTP protocol.

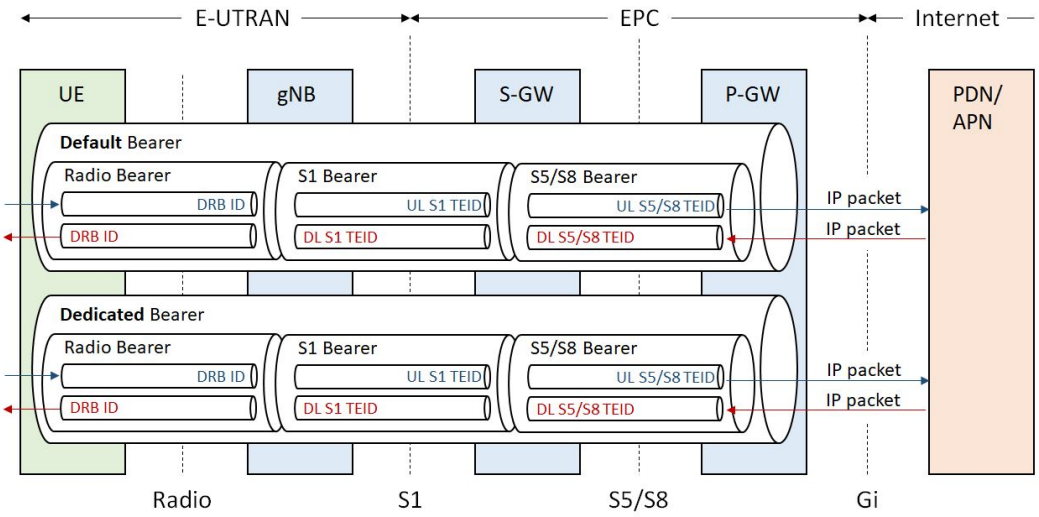

Fig. 2.4 EPS Bearer Service Architecture (based on [58])

2.3.4 GPRS Tunneling Protocol (GTP)

GTP is a tunneling protocol defined by the 3GPP used to carry encapsulated user packets and signalling messages in the LTE network [57]. Its main idea is to encapsulate a packet (user or signaling packet) by adding an outer IP header and a GTP header to the original packet. This mechanism of tunneling helps, in one side, the user to keep its assigned IP address when it moves from one place to another, and on the other side, to see the PGW as the first IP hop once the latter decapsulates the GTP packet and sends it to the corresponding PDN. GTP consists of the following types:

- **GTP Control Plane (GTP-C):** used to carry signalling messages in the interfaces $S11$ between MME and SGW, and $S5$ between SGW and PGW, such as activating deactivating subscribers sessions, mobility and QoS management.
- **GTP User Plane (GTP-U):** used to carry encapsulated user data packets in the interfaces $S1$ between gNB and SGW, and $S5$ between SGW and PGW.
- **GTP Prime (GTP'):** used to carry charging data.

As the GTP-U transports the user data packets, it corresponds to the implementation of the S1 and S5/S8 Bearers, where each Bearer consists of two UpLink and DownLink tunnels while each tunnel is identified by a unique 32 bit Tunnel Endpoint Identifier (TEID), witch referred to as Uplink (UL) TEID and Downlink (DL) TEID respectively and used to identify the corresponding Bearer, as shown in figure 2.4.

Once the UE attached to the network, its data is transported separately through the UpLink towards the PDN and the DownLink from the PDN. To this end, the PGW, SGW and gNB create respectively UL S5/S8 TEID, UL S1 TEID and DRB ID and send them respectively to SGW, gNB

and UE to transport the user packet in the UpLink direction. In the same way, the gNB and SGW create respectively DL S5/S8 TEID and DL S1 TEID and send them respectively to SGW and PGW to transport the user packet in the DownLink direction.

2.4 Software Defined Networking (SDN)

2.4.1 History of Programmable Networks

The decoupling of the control and data planes, and the programmability on the control plane have been proposed as a way to facilitate network evolution. In particular, SDN is an emerging networking architecture, that decouples the network intelligence from the network devices, enabling thus more flexibility and simplicity to design, operate, configure and monitor, by allowing software developers to deploy services and features in the network by programming on controllers.

Decoupling of control and data planes, and programmable networks were taken long ago [60][90][115]. In the mid 1990s, Active Networks [149] [148] was the early approaches proposed for enabling the programmability of network infrastructure for customized services.

DCAN [102], Open Signaling [39], ForCES [162], 4D Project [67], NCP [140], Tempest [103], RCP [38] are the earliest initiatives on separating data and control planes. In the mid 1990s, the DCAN project was proposed to design and develop the necessary infrastructure to improve the management of ATM networks. The focus was the separation of the control and management functions from the devices (i.e., the ATM switches) to be delegated to external entities. In 2003, the Internet Engineering Task Force (IETF) released the Forwarding and Control Element Separation (ForCES) framework, which makes use of an API to separate the control and packet forwarding elements. In 2004, the 4D Project as well supports the idea of centralized management by decoupling the routing decision logic from the protocols dedicated for the interaction between network devices. The Path Computation Element (PCE) architecture [13] was presented in 2006 to improve management in multi-protocol label switching (MPLS) and Generalized MPLS networks. In 2007, the Ethane (i.e., SANE) project was proposed [40], which represents the immediate predecessor of the OpenFlow protocol [101], where a centralized controller is responsible for routing decisions in top of simple flow-based Ethernet switches.

2.4.2 SDN Benefits and Challenges

With the separation of the control and data planes, the control is logically centralized in a dedicated entity controller while network switches become simple forwarding devices. This brings potential benefits of enhanced configuration, improved performance, and encouraged innovation in network architecture and operations [157].

- **enhanced configuration** with the heterogeneity of current networks design adding new network elements is challenging and involves a certain level of manual processing. SDN is well placed to

cope with such challenges by automatically configure divers network devices from a centralized controller.

- **improved performance** in the traditional networks design, the coexistence of heterogeneous technologies, and the lack of global view makes it difficult to improve the infrastructure use as well as the QoS. SDN will help to remedy with such a situation, by having a global view with more accuracy over all the data plane devices.

- **encouraged innovation** the programmability in SDN encourages innovation by designing and implementing new ideas in the separated controller without any need to modify the data plane devices.

Despite the benefits of SDN, the technology is still maturing and introduces new challenges [90][157], including standardization and interoperability, scalability and resilience as the most urgent ones:

- **standardization and interoperability** SDN is still suffering from the immaturity or the lacking of standardization such as the missing of a standard north-bound API or a high level programming language for SDN application development, service on boarding and orchestration as well as network control are not yet fully standardized. Moreover, it suffers from the lack of interoperability with traditional networks.

- **scalability** in SDN networks, the control plane needs to be able to process millions of flows per second without compromising the quality of its service. However, this could increase the network load which could be a potential bottleneck. Furthermore, an extra latency can be introduced by the flow setup process, where the flow tables of devices must be configured in real time.

- **resilience** relocating the SDN control plane functionality may compromise the communications between the control and data planes and result in the loss of availability, in particular the critical control plane functions such as those related to link failure detection, congestion and traffic load balancing.

2.4.3 SDN Architecture

As suggested by ONF [116] in Figure 2.5, a logical view of the SDN architecture can be depicted as a composition of different layers, namely an infrastructure layer, a control layer, and an application layer, each layer has its own specific functions.

Instead of enforcing policies, understanding and processing thousands of protocol standards, SDN greatly reduces the network devices in the infrastructure layer (e.g. switches, routers, etc.), to dumb devices, that only perform the forwarding action. On the other hand, the control logic and network intelligence are moved to the controllers or Network Operating System (NOS) in the control layer,

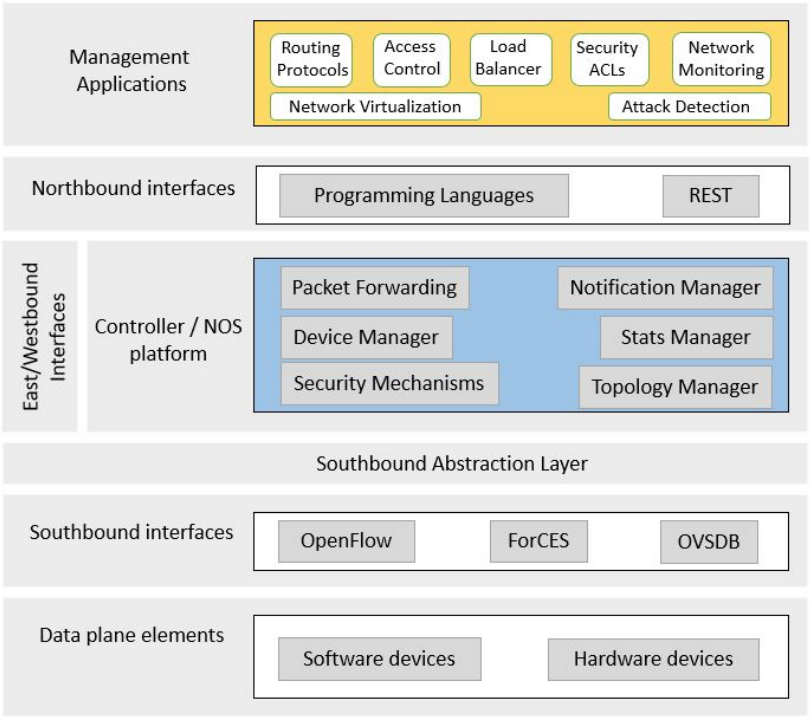

Fig. 2.5 Detailed SDN Architecture (based on [90])

specifying through the southbound interface a set of functions to be performed by the data plane. The control layer provides the application layer access to services and applications through the northbound interface. In what follows, we introduce each layer.

Infrastructure layer

The infrastructure layer consists of a set of simple forwarding switching devices (switches, routers, middlebox appliances, etc.), with no embedded intelligence to take decisions and responsible only for processing packets as well as gathering and reporting network status based on pr-installed forwarding rules. In comparison with conventional networking equipments, the network intelligence is logically centralized to an external entity, namely, SDN controller or NOS.

The SDN switching device stores the received rules from the controller, including packet forwarding rules, in its local memory such as Static Random Access Memory (SRAM), Ternary Content Addressable Memory (TCAM) [79]. Then, based on these rules, it performs the packet forwarding. Due to the limited capacity of the onboard memory (i.e., TCAM, SRAM) in the switching devices, SDN helps introducing several approaches such as route aggregation or summarization and proper cache replacement policy. Moreover, unlike the legacy networks where the packet forwarding is generally based on IP or MAC addresses, the SDN packet forwarding is extended to additional parameters such as Virtual Local Area Network (VLAN) tag, TCP or UDP port, tunnel identifier and ingress switch port to forward a packet.

Software switches as well are now used extensively in data centers and virtualized network infrastructures. In this context, several solutions have been implemented to support SDN architecture and OpenFlow protocol, such as Xorplus [14], OVS [120], [88], ofsoftswitch13 [54], OpenFlow Reference [123], Switch Light [28] and Pantou [160].

Southbound interfaces or APIs are the crucial instrument connecting the control and data planes, and plays in important role in introducing and accepting any new technology based on SDN architecture. In SDN, one of the most accepted standard for the southbound interface is OpenFlow, due to its flexibility in the network usage and operation as well as the ability to operate switches (software, hardware) and routers from divers vendors. Several APIs for southbound interfaces have also been proposed in SDN, such as OVS Database (OVSDB) [23], ForCES [162], OpenState [27], OpFlex [99], hardware abstraction layer (HAL) [125], programmable abstraction of data path (PAD) [25], revised open-flow library (ROFL) [144] and POF [142].

In our work, we opted for OVS, which is an SDN based multi-layer open source virtual switch targeted at virtualized environments, introduced the first time by Pfaff et al. in [127]. It was designed to enable network automation through programmability, while supporting standard management interfaces and protocols such as OpenFlow. OVS is included in the Linux kernel since version 3.3 and licensed under the open source Apache 2 license [121].

Control layer

The control layer bridges the infrastructure layer and the application layer and consists of a set of NOS or controllers, representing the core component in the SDN architecture, where the intelligence and complexity reside. With NOSs in SDN, there is no need to care about the low-level details for each device separately and networking functionalities such as network topology information, network state, device discovery, and network distribution, will be moved to the NOS component. In the following paragraphs, we describe the main components corresponding to the control layer: the northbound interface and the core controller.

Figure 2.5 depicts the common elements extracted from existing control platforms such as ONOS [26], NOX [68], ForCES [162] and OpenDaylight [122]. In general NOS platforms, the control layer translates SDN application requirements in the application layer into packet forwarding rules and policies and provides them with a synchronized global view, through the northbound interface, such as high-level programming languages and Representational State Transfer (REST) APIs [135]. The high-level languages for SDN controllers are classified in two categories [157]. The first one is characterized by special features designed to improve the network behavior control for SDN, such as FML [73], Frenetic [62], Nettle [153]. The second category utilizes the Software Development Kit (SDK) provided with libraries for desirable networking features, in existing mature languages such as Python, C++ and Java.

The SDN controller or NOS consists of a set of services such as Packet Forwarding, Notification Manager, Statistics Manager, Device Manager, Topology Manager and Security Mechanisms, as

illustrated in Figure 2.5. The logical operating mechanism of these services can be described based on the upward and downward interacting with the application and infrastructure layers respectively. In the downward direction, the controller translates the applications requirements into networking rules and configurations in the corresponding services, such as the transformation of routing application policies into packet forwarding rules in the Packet Forwarding service. The upward direction is related to collecting statistics, global network status from the infrastructure for network monitoring and decision making.

In our work, we opted for ONOS as SDN controller. In fact, ONOS is the leading open source SDN controller that is firstly released by Open Networking Lab [117], [119]. ONOS is written completely based on Java as Open Services Gateway initiative (OSGi) bundles and modules that can be installed and started dynamically in a single Java Virtual Machine (JVM), which allows ONOS to be running in different Operating System (OS) platforms.

Application layer

As depicted in figure 2.5, the application layer consists of applications in a wide variety of networked environments and interacts with the services in the control layer via the northbound interface (i.e., the API). By having a global view with instantaneous status on the underlying physical networks as well as the programmability using the high programming languages, these applications can dynamically change the network behavior. SDN applications can be grouped [157], [90], [115] in several areas such as Routing and TE, Green Networking, Data Center Networking, Network Virtualization and Network Security.

In particular, TE refers to the process of improving the QoS and user experience, network performances, by efficient use of resources, analyzing network traffic, optimizing traffic routing and load balancing. In SDN the centralized view, monitoring capabilities as well as the programmability features pave the way for new TE techniques and better traffic control and management. On the other hand, Green Networking refers to reduction of unnecessary energy consumption in the networking domain, where data-centers and networking infrastructure rely on under-utilized energy-consuming powerful devices. SDN has gained attention as a suitable solution to address the QoS, congestion, energy consumption, flexibility and resiliency issues in data centers networks because of its centralized control and programmability features.

2.4.4 OpenFlow

The OpenFlow protocol is the driving force behind the wide-spread adoption of SDN and its implementation in both hardware and software [116]. OpenFlow is the first standard interface designed specifically for SDN, which structures and standardizes communication between the control and data planes and allows applications to dynamically program the flow table of different data plane devices. It was developed, firstly, as an academic project at the Standford University in 2008 [101]. Then, in 2011, the standardization of OpenFlow specification was moved to ONF [117]. The OpenFlow

specification has evolved through several releases [118], each with new features. The first release of OpenFlow, version 1.0, was published on December 31, 2009, and the last version 1.5 which was released on March 2015.

In the OpenFlow architecture, illustrated in figure 2.6 [118], the OpenFlow switch, communicates with the external controller through one or more OpenFlow channels and forwards packets based on a pipeline of flow tables where each entry of a flow table contains a set of flow entries: 1) a matching rule 2) instructions to apply to matching packets, 3) counters that keep statistics of particular matching packets/flow, such as number of received packets, number of bytes and duration of the flow, 4) priority that determines the flow rule to be selected when a packet is matched against several flow entries in the flow table, where higher priority rules are installed above lower priority ones, 5) timeouts corresponds to the maximum time (i.e., hard time-out) or idle time before flow entry is removed from the flow table, 6) cookie enables OpenFlow to filter flows selected for flow modification and deletion requests, and 7) flags used to change the management of flow entries in the flow table.

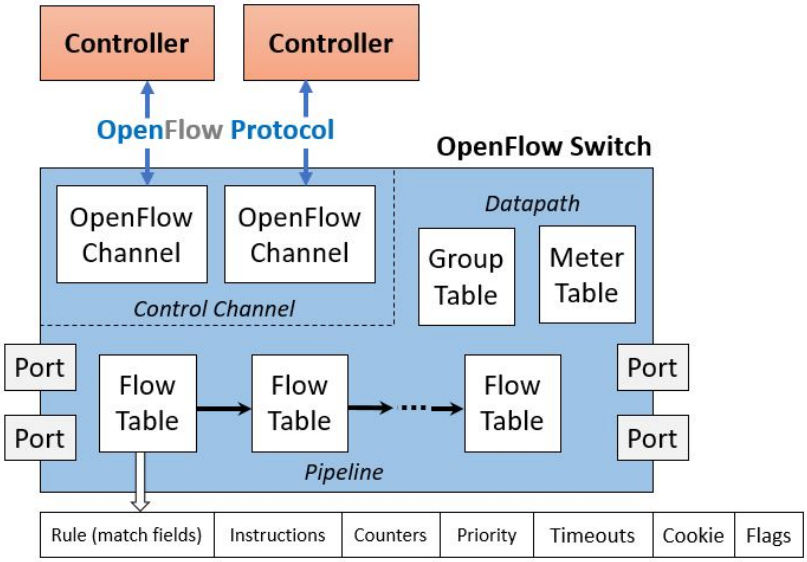

Fig. 2.6 OpenFlow Architecture (based on [118])

Upon the arrival of new packets at an OpenFlow switch, the lookup process starts in the first flow table and may continue to additional flow tables of the pipeline and ends either with a match in which the switch applies the appropriate set of instructions, or actions, associated with the matched flow entry or with a miss if no match is found in the flow tables pipeline, where the action will depend on the instructions defined by the table-miss flow entry. Instructions in each flow entry consists of a set of actions that determine packet forwarding, packet modification and group table processing as well as describe the pipeline processing when a packet needs to be sent to subsequent tables for further processing. One of the actions is to send packets to Group Tables for flooding, and more complex forwarding such as multi-path, fast reroute, and link aggregation.

OpenFlow messages

The controller exchanges control messages with OpenFlow switch through the OpenFlow channel that is usually encrypted by using Transport Layer Security (TLS). These messages can be categorised in three types, each with their own setup of sub-types [118]: controller-to-switch, asynchronous, and symmetric. The message types used by OpenFlow are described as follows:

- **controller-to-switch messages** used by the controller to directly manage and inspect the OpenFlow switch.
- **asynchronous messages** are initiated by the switch without a solicitation from the controller, to denote network events such as packet arrival or switch state change.
- **symmetric messages** are triggered either by the switch or the controller and sent without solicitation.

2.4.5 Distributed SDN Architecture

In SDN, the control plan can be logically centralized, by having a global view for the entire network, while physically centralized in a single controller or distributed in multiple controller nodes. Even the physically centralized control plane gives more simplicity for the network design, studies on several centralized controllers such as NOX [68], Floodlight [61] and Beacon [56] have revealed several issues in terms of:

- **robustness and vulnerability:** using a single controller to manage all forwarding devices of the network can be a single point of failure and bottlenecks for the whole network, where a failure in the controller entails failure in the entire network. Moreover, attacking on control plane communications such as Denial-of-Service (DoS) attacks, can compromise the controller–device link and cause failure in the whole network [89].
- **scalability and reliability:** in large scale networks such as Data Centers and Service Provider Networks, the control of thousands of switching elements with millions of flows and events to process per second can be challenging which may involve high latencies and can generate overhead and bandwidth degradation [21].

Contrary to a physically centralized controller, maintaining the logical centralized control plan over a physically distributed NOS or controllers, is well placed to cope with the challenges of centralized SDN architectures mentioned above (i.e., scalability, reliability, robustness and vulnerability) from small to large-scale networks, as demonstrated in several distributed controllers such as Onix [87], ONOS [26], Hyperflow [150], Kandoo [71]. As in the distributed design, the controllers can be spread across the network, dealing thus with the single point of failure issue by assigning devices of the failed controller to a neighbor controller. Moreover, the scalability and resilience can be improved while reducing latency since each controller node will manage a specific network segment.

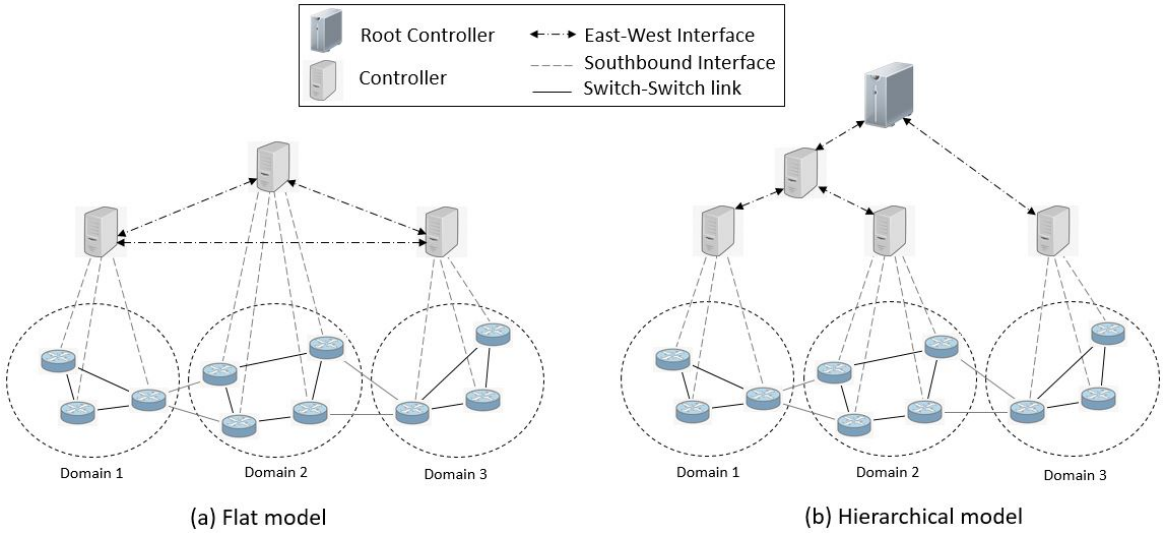

Fig. 2.7 Distributed SDN Architectures (based on [78])

In the physically distributed design of SDN controllers, two main topology-related architectures can be identified [78]: i) a flat SDN controller design, in which the network is partitioned horizontally into different controller domains, where each domain is handled by a single controller in charge of managing a disjoint subset of SDN-enabled switches, as depicted in figure 2.7 (a). ONOS [26], Onix [87] and Hyperflow [150] are examples of distributed SDN controllers supporting the flat design, and ii) a hierarchical SDN controller design, in which, the network is partitioned vertically into different controller domains, based on the locality requirements, where applications with frequent events may be handled by local controllers close to the switch, as shown in figure 2.7 (b). Kandoo [71] is an example of distributed SDN controllers supporting a hierarchical design.

2.5 Machine Learning Techniques Applied to Software Defined Networking (SDN)

2.5.1 Knowledge Defined Networking (KDN)

The concept of KP, was originally introduced by D. Clark et al. in [45]. Which corresponds to a new network design that embodies cognitive system and ML to self-driving network, by introducing network automation and providing knowledge to decide on how to operate the network. Contrary to traditional networks, where this concept can be challenging due to the partial vision of the network as well as the lack of monitoring data, SDN separates the control and data planes which, in one side, provides a global view to the KP, and on the other side, paves the way to new network monitoring techniques (i.e., network telemetry) that dynamically provides the KP with information generated in the network and collected to an external NA platform. In this way, the combination of a KP to the

conventional SDN paradigm as well as network telemetry, and NA unveils a new paradigm called KDN [104].

Figure 2.8 shows the overall architecture of KDN [45],[104], in which two main plains are added to the SDN architecture: management plane and knowledge plane. These functional planes can be described as follows:

- **management plane:** takes as input data collected directly from the data plane such as packet-level network state using In-band Network Telemetry (INT) [85] or through controller modules such as network topology and devices configuration, to ensure network operation and performance by acting directly on the controller or providing richer global view from these information to ML modules in the Knowledge plane to take further decisions on how to operate the network.
- **knowledge plane:** takes advantages of the rich view provided by both the control and management planes to learns the behavior of the network by processing the NAs collected by the management plane and converts them to the form of knowledge via ML modules, which enables automatic decisions or recommends the best actions for human intervention on how to operate the network.

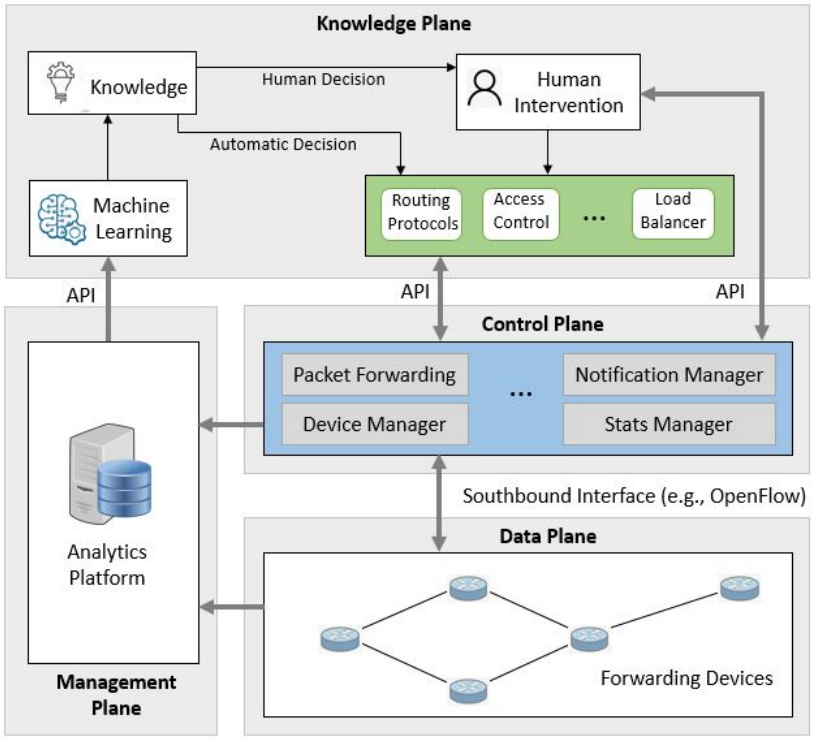

Fig. 2.8 Knowledge Defined Networking (KDN) Architecture (based on [45][104])

2.5.2 Machine Learning Paradigms

Usually, two main phases are needed to build a machine learning model, which are mainly: the training phase and the decision making phase [158]. After data preparation, the data is usually partitioned into a training set and a test set. In this way, the machine learning algorithm learns the system model from the training data, in the training phase, and uses the test data to evaluate the learning. In the decision making phase, new inputs can be presented to the trained model and the output is the estimated value.

ML methods can be classified according to different criteria [167],[31]. Based on the parameters of the model, the ML methods can be classified in two categories: a linear model which is characterised by a linear combination of features and nonlinear model which includes Deep Learning and traditional models. According to task type, the ML can be classified into four categories: classification, clustering, regression and rule extraction. The classification problems identify the category of new observation. Whereas, clustering problems group similar observations together. The regression problems estimates the relationship between a set of new input observations and a set of discrete or continuous observations. Rule extraction problems determine statistical relationships in data.

Based on the training method, ML algorithms are basically classified into four categories [31]: the supervised learning, the unsupervised learning, the semi-supervised learning and reinforcement learning. These categories can be described as follows:

- **supervised Learning:** creates models based on labeled training datasets, consisting of a set of training examples (i.e., inputs and known outputs). This learning technique is used to infer patterns or behaviors in the “known” training datasets. In this way, the trained model can be used to estimate “unknown” outputs for new inputs. Typically, all regression and classification algorithms are supervised learning methods. Supervised learning algorithms are widely used in many SDN applications. We will focus on those used, in our work, for traffic prediction, such as LR, ARIMA and LSTM.

- **unsupervised Learning:** Contrary to supervised learning, the unsupervised learning algorithm is given unlabeled training datasets to create models that can reveal discrimination between patterns in the sample data and thus cluster them into different groups based on these patterns. In this approach, the most commonly used method is "clustering" such as the well-known clustering algorithm K-means [65].

- **semi-supervised Learning:** is a learning method that combines supervised learning (with only labeled data) with unsupervised learning (with no labeled data). In many practical problems, it is difficult to acquire a large amount of labeled data comparing to unlabeled data. Moreover, the performance of the trained model can be improved while using unlabeled data during the training. Which lead to the development of semi-supervised learning approach that makes the best use of both labeled and unlabeled data. In SDN applications, the traffic from unknown applications can be handled using semi-supervised approach.

- **reinforcement Learning:** contrary to the previous approaches, where the learning is based on examples in the training dataset, reinforcement learning is a technique based on an agent that interacts with its environment to learn the best action to maximize a numerical reward. The objective is to train an agent based on a set of state-action pairs and rewards in order to find a suitable action model or to learn the best policy (i.e. sequence of actions) that would maximize the cumulative reward of the agent. Typically, DRL can be classified in this category.

2.5.3 Linear Regression (LR)

The LR is a largely statistical prediction method that attempts to model the relationship between an explanatory independent variables and a dependent variable, by fitting a linear equation to observed data, where the objective is to predict the outcome of the dependent variable from the explanatory independent variables [82]. Equation (2.1) shows the LR model:

$$\widehat{y_i} = \lambda_0 x_{i0} + \lambda_1 x_{i1} + \lambda_2 x_{i2} + ... + \lambda_p x_{ip} + \varepsilon_i = \sum_{j=0}^{p} \lambda_j x_{ij} + \varepsilon_i \tag{2.1}$$

Where $x_j = [x_{0j}, x_{1j}, ..., x_{nj}]$ are the explanatory variables, $\widehat{y} = [\widehat{y_0}, \widehat{y_1}, ..., \widehat{y_n}]$ is the dependent variable, λ_j are the regressors that determine the model and ε_i is the error of the model which expresses the missing information in the model.

One of the most used methods to estimate the regressors (i.e., the λ_j parameters) in a LR model called Ordinary List Squares method (OLS) [155]. Its main idea is to minimize the sum of the squares of the differences between the predicted variables (i.e., dependent variables (i.e., $\widehat{y_i}$) and the explanatory variables (i.e., x_j). Equation (2.1) can be written in a matrix form as follows:

$$X\lambda = Y \tag{2.2}$$

Where

$$X = \begin{bmatrix} x_{1,0} & x_{1,1} & \cdots & x_{1,p} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,0} & x_{n,1} & \cdots & x_{n,p} \end{bmatrix}, \lambda = \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_p \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

The regressor or coefficient vector λ can be determined based on the OLS method as shown in equation (2.3), where X^T denotes the matrix transpose of X and the (-1) denotes the matrix inverse.

$$\lambda = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathrm{Y} \tag{2.3}$$

If the matrix X consists of only two explanatory variables (x_{i0}, x_{i1}), then, the corresponding model refers to *simple regression model*, which can be represented as follows:

$$\widehat{y_i} = \lambda_0 + \lambda_1 x_i + \varepsilon_i \tag{2.4}$$

The coefficient vector λ can be determined in equation (2.5), where *Cov* and *Var* denote respectively the covariance and the variance.

$$\lambda = \frac{\sum x_i y_i - \frac{1}{n}\sum x_i \sum y_i}{\sum x_i^2 - \frac{1}{n}(\sum x_i)^2} = \frac{Cov(x,y)}{Var(x)} \tag{2.5}$$

The LR can be interpreted geometrically as finding the line or the plane that best fits the set of (x_i, y_i) points, by minimizing the sum of the squared distances between each data point.

2.5.4 Auto-Regressive Integrated Moving Average (ARIMA)

ARIMA is a time series forecasting model, which attempts to identify the behavior of time series, based on its own past values (i.e., its own lags and the lagged forecast errors), in order to predict or forecast the future evolution [37]. ARIMA is extending the Auto-Regressive Moving Average (ARMA) model to support the non-stationarity in the evolution of data.

In the ARIMA model, the Integrated (I) part represents the number of differentiation in order to make the series stationary. On the other hand, the Auto-Regressive (AR) part represents the relationship between an evolving variable Y_t and its own lagged values in previous time intervals $Y_{t-1}, Y_{t-2}, ..., Y_{t-p}$, as shown in equation (2.6), where, Y_{t-i} are the lags of the time series, ϕ_i are the coefficients of lags that the model estimates and c is the intercept term.

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + ... + \phi_p Y_{t-p} \tag{2.6}$$

The Moving Average (MA) part represents the relationship between an evolving variable Y_t and its lagged forecast errors $e_t, e_{t-1}, e_{t-2}, ..., e_{t-q}$ which are the errors of the AR models, as shown in equation (2.7), where θ_j are the coefficients of error terms.

$$Y_t = c + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_q e_{t-q} \tag{2.7}$$

The ARIMA model can be represented by combining AR and MA terms as shown in the following equation:

$$Y_t = c + \sum_{i=1}^{p} \phi_i Y_{t-i} + \sum_{j=0}^{q} \theta_j e_{t-j} \tag{2.8}$$

As shown in (2.8), the ARIMA model is characterized by the three parameters (p, d, q) representing respectively the number of AR terms (i.e., p), the degree of differentiation needed for stationarity (i.e., d) and the number of lagged forecast errors (i.e., q).

In order to determine the order of differencing d , the Augmented Dickey Fuller (ADF) test can be used [110]. The idea is to keep difference the time series and increment d while there is no stationarity. On the other hand, the required number of AR terms p can be determined by inspecting the Partial

Autocorrelation Function (PACF) plot [37]. Similarly to the PACF plot, the PACF plot can be used to determine the number of MA terms q.

2.5.5 Long Short Term Memory (LSTM)

LSTM is a variation of Recursive Neural Network (RNN), in which the output from a step is fed as input of the next step. It was designed to tackle the problem of long-term dependencies of RNN by preserving the error that can be back-propagated through time and layers [74]. It consists of Memory blocks containing memory cells that store the temporal state of the network C_t, while controlling the flow of information by a special multiplicative unit called Operation Gates which are mainly Forget Gate, Input Gate and Output Gate, as shown in figure 2.9.

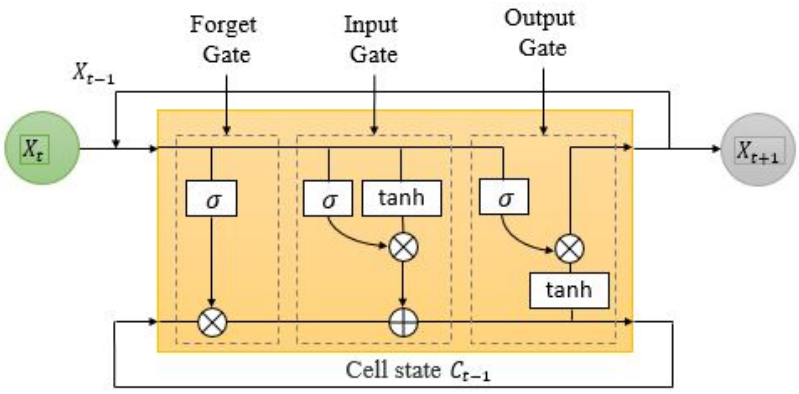

Fig. 2.9 LSTM node (based on [74])

The Forget Gate, as shown in equation (2.9), decides what information to remove from the cell state C_{t-1} based on the Sigmoid function (σ), it looks at: 1) the concatenation of the input X_t and the previous activation output X_{t-1}, 2) the bias b_f, and outputs a vector f_t of values between 0 and 1, whose values that are closer to 0 mean forgetting the information that matches in memory C_{t-1}.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{2.9}$$

The aim of the Input Gate is to propose new information $\tilde{C}_t$ (2.10), and to choose which information to put in the memory C_{t-1}, by using i_t in (2.11).

$$\tilde{C}_t = tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{2.10}$$

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{2.11}$$

Based on the Forget and Input Gates, the memory is updated as follows:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.12}$$

Finally, the Output Gate aims at firstly pushing the memory C_t values in $\{-1,1\}$ by using *tanh* function in (2.14), then it decides which part of the memory C_t will be in the output by using the Sigmoid function in (2.13).

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{2.13}$$

$$X_{t+1} = o_t * tanh(C_t) \tag{2.14}$$

Note that the Sigmoid σ and *tanh* functions are defined as follows :

$$\sigma(x) = \frac{1}{1+e^{-x}}, tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$$

2.5.6 K-means

The K-means is the most commonly used unsupervised learning algorithm for clustering, which is widely used in network partition problems [18]. Its objective is to identify a set of unlabeled data into different clusters. The K-means algorithm takes as input only two parameters: 1) the initial dataset, 2) the number of clusters k, and outputs the set of k clusters with their corresponding elements.

The K-means algorithm works as follows [65]: 1) select p random points as cluster centers called *Centroids*, 2) assign each point to the closest cluster based on the Euclidean norm, 3) recalculate the centroid for each cluster by computing the average of the assigned points, 4) repeat steps 2 and 3 until none of the cluster assignments change.

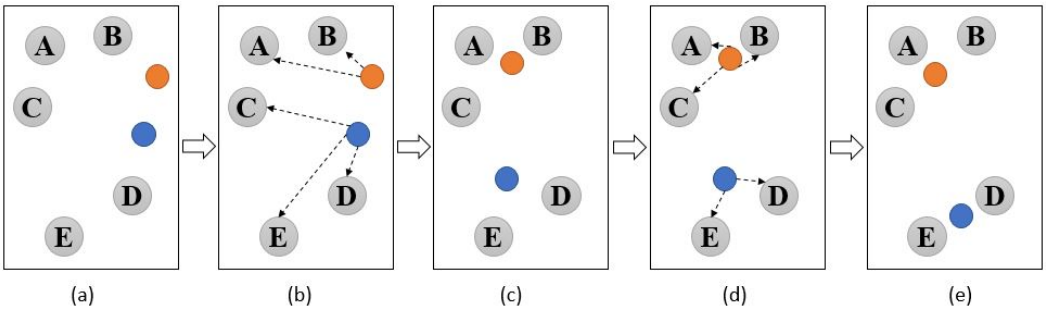

Fig. 2.10 Example of K-means algorithm (based on [158])

Figure 2.10 shows an example of the K-means algorithm. Using the number of clusters $k = 2$, initially two points have been chosen randomly as centroids in (a), then in (b) assigning or labeling each node with closest centroid chosen in (a), thereafter, measuring the Euclidean centroids in (c). In (d) assignment of the nodes based on the new centroids, and finally the algorithm converges in (e), where none of the cluster assignments change.

2.5.7 Deep Reinforcement Learning (DRL)

The machine learning technique *QL*-earning (*QL*) basically based on an autonomous agent that interacts with the environment by sequentially taking actions while maximizing cumulative rewards

[106][107]. As shown in figure 2.11, this can be described as MDP in which the next state s_{i+1} depends on the current state s_i and the selected action by the agent according to a specific state transition probability distribution $P(s_i, a_i, s_{i+1})$, which represents the probability of switching to state s_{i+1} after action a_i in state s_i. In QL, the agent senses the environment by identifying the current state s_i and then selects the action a_i to execute. The environment subsequently feeds back a reward to the agent while updating the current state to the new state. Then, the trajectory of states, actions and rewards constitute a MDP: $s_0, a_0, r_0, s_1, a_1, r_1 ...$ The objective is to learn the best policy $\pi(a_i|s_i)$ while maximizing the cumulative rewards of the current and next states:

$$R = \sum_{i=0}^{n} \varphi^i r_{i+1} \tag{2.15}$$

Where $\varphi \in [0,1]$ is a factor to discount future rewards. Given a policy π, the expectation of accumulated rewards from action a_i in a state s_i can be estimated by the the Q-value function $Q_\pi(s_i, a_i) = E[R|s_i, a_i, \pi]$. Thereafter, the best policy corresponds to the highest Q-value in each state: $Q^*(s_i, a_i) = max_\pi Q_\pi(s_i, a_i)$, this function can be defined recursively according to the Bellman equation as follows:

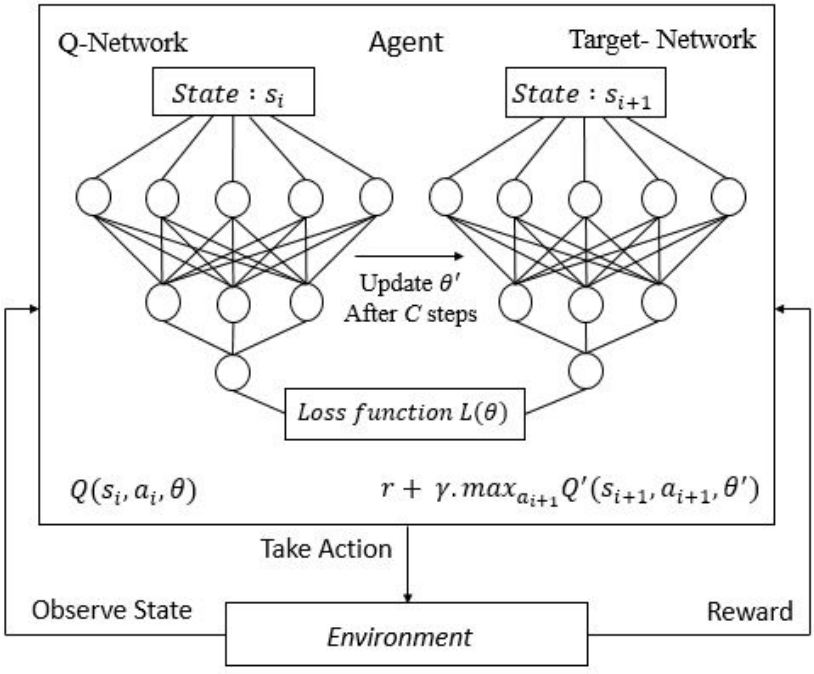

Fig. 2.11 Deep Q-Network (DQN) Model (based on [106][107])

$$Q_\pi(s_i, a_i) = r_i + \varphi . Q^*(s_{i+1}, a_{i+1}) \tag{2.16}$$

The policy π can be improved by dynamically updating the $Q_\pi(s_i, a_i)$ as follows:

$$Q_\pi(s_i, a_i) \leftarrow Q_\pi(s_i, a_i) + \eta.\Delta \tag{2.17}$$

$$\Delta = r_i + \varphi.max_{a_{i+1}} Q_\pi(s_{i+1}, a_{i+1}) - Q_\pi(s_i, a_i) \tag{2.18}$$

Where $\eta \in [0,1]$ is the learning rate, and the temporal difference (TD) error Δ corresponds to the correction for the Q-value estimation. The QL technique stores and updates the Q-values in look-up tables, which makes it slow to reach the best policy when exploring the entire table when the number of possible states becomes very large, which affects significantly the performance of Q-learning. To cope with this challenge the DQN makes use of NN to approximate the estimation of the Q-value function. The DQN networks take as input the state vector and the output is a vector of action Q-values, and its corresponding Loss function is constructed based on the mean square deviation defined as follows :

$$L(\theta_i) = (TargetQ - Q(s_i, a_i, \theta_i))^2 \tag{2.19}$$

$$TargetQ = r_i + \varphi.max_{a_{i+1}} Q(s_{i+1}, a_{i+1}, \theta_i) \tag{2.20}$$

Where θ_i is the network parameter at iteration i. The convergence of the Loss function $L(\theta_i)$ is not stable when using only one NN. To improve the convergence stability, DQN adopts the method called *Experience Replay*, which corresponds to creating two NNs, that have the same architecture, with parameters θ and $\theta^{'}$. The first one is used to retrieve Q-values while the second one includes all updates in the training. After C steps the target network parameters θ' are updated. This mechanism is illustrated in Figure 2.11.

2.6 Conclusion

This chapter overviewed the main technologies used in this thesis namely 5G mobile networks, SDN and ML. Firstly, we gave an overview of the 5G requirements and challenges and the 5G detailed architecture used in this work. Thereafter, we overviewed the main enabler SDN, the OpenFlow protocol and distributed SDN architecture. Finally, we gave a brief overview of the main ML learning techniques used in this thesis for traffic prediction and optimisation namely LR, ARIMA, LSTM, K-means and DRL.

Traffic prediction and optimisation, SDN controller placement and Network Slicing sill involve a number of challenges that we aim to address in this thesis. As a first step towards this, in the next chapter, we will review the state-of-the-art related to Data-driven Dynamic Optimization of Traffic Workload and Network Slicing for 5G Networks and beyond.

Chapter 3

Data-driven Dynamic Optimization of Traffic Workload and Network Slicing for 5G Networks and beyond - State of the Art

Contents

3.1 Introduction

In this chapter, we will provide an overview of the approaches found in the literature regarding our three axes of research identified in this thesis. We will start by reviewing the network traffic monitoring approaches based on SDN and OpenFlow protocol as accurate traffic monitoring is the key challenge to optimize QoS. Then, we will discuss the ML based QoS aware routing approaches. Next, we will discuss different approaches on E2E network slicing. In particular, we will focus on the 5G architecture, IoT platform and transport network used in the context of ML. Finally, we will discuss different approaches on SDN controller placement. Particularly, we will review strategies proposed for SDN control and data plane clustering.

3.2 Research Axes: State of the Art

3.2.1 Deep Q-Network and Traffic Prediction based Routing Optimization in Software Defined Networks

In the following, we first survey the literature on traffic monitoring approaches to the collection of data plane statistics in SDN. Then, we focus on the ML techniques used for traffic prediction and routing optimization.

Traffic Monitoring

Recently, SDN [116] with OpenFlow protocol [101] implementation is getting a lot of attention. There are many OpenFlow software implementing the SDN architecture, where the most used in the control plane ONOS [26], POX [81], Ryu [8], and OVS [120] in the data plane. In traditional networks, many monitoring tools are available. SNMP [15], Cisco NetFlow [6], sFlow [10]. However, these monitoring tools are not compatible with the OpenFlow network. Several works have been proposed in OpenFlow monitoring. Authors in [44] proposed PayLess, a network monitoring framework for SDN, which is built on top of an OpenFlow controller's northbound API and provides a high-level RESTful API, and offers an adaptive scheduling algorithm for polling, which achieves the same level of accuracy as continuous switch polling with much less communication overhead. In [152], authors proposed OpenNetMon, a network monitoring tool based on OpenFlow, that collects statistics such as throughput and packet loss from the edge devices in order to improve the measurement accuracy and reduce the computation overhead. PathMon [105] provides a way to collect per-flow statistics such as throughput and packet loss by inserting a separate set of flow entries called monitoring entries into every switch along a path to be monitored. In these works, statistics can be collected proactively by using $FlowStatisticsRequest$ message or reactively by using the $FlowRemoved$ notification massage. However, they did not provide mechanisms to measure latency by using only the OpenFlow protocol.

There have been many studies on latency measurement in SDN using OpenFlow protocol [20][164]. The most relevant work is described in [164] in which the authors present SLAM, a

framework for Software-defined Latency Monitoring between any two network switches. The delay is measured inside the network by capturing directly path information from network devices. Instead of measuring latency for a specific path as described in these works, we propose to measure metrics, such as latency throughput per-flow size, for all critical links in the network, where the probability of congestion is important. In this way, an important number of paths can be covered without affecting performances.

Machine Learning based QoS-aware Routing

Optimizing flow routing in SDN-based networks is crucial to meet the needs for efficient resource allocation. For that reason, a wide range of solutions have been proposed in the literature.

In [134], authors propose approaches based on genetic and ant-colony algorithms to optimize flow rule placement. However, these approaches are limited to certain situations and do not cover more complicated routing problems. Several research works have been made to install flow rules across pre-computed optimal paths, by exploiting the SDN controller's global visibility. In [70], authors propose an approach to reduce power consumption and network congestion, where they compute the optimal topology that can accommodate the expected traffic demands, then a traffic load balancing by distributing flows across k-shortest paths of optimal topology, by resolving an Integer Linear Program (ILP) Problem.

Holt–Winters (HW), ARMA/ARIMA models [46] are traditional linear prediction methods, that are widely used for network traffic forecasting. Works in [24] and [22] both deploy NN based on SDN for network Traffic Matrix (TM) prediction. Authors in [24] presented an approach to predict aggregated Ethernet traffic with NNs employing multiresolution learning (MRL). They consider the problem of forecasting the transfer rate, by predicting future values, given a set of transfer rates observed on a specific link. The experimental results show that nonlinear traffic prediction based on NNs outperforms linear forecasting models (e.g. ARMA, Auto-regressive Auto-regressive (ARAR), HW) which cannot meet the accuracy requirements. In [22], authors present a LSTM based RNN framework which makes use of model parameters to train prediction models for large scale TM prediction, by training and comparing LSTM models at various numbers of parameters and configurations. Simulations show that the proposed LSTM models converge quickly and achieves high prediction accuracy in a few seconds of computation.

Authors in [83] used Reinforcement Learning techniques in the context of routing optimization, where they present a Q-Routing algorithm discovering efficient routing policies in dynamically changing networks, without having to know in advance the network topology and traffic patterns. As in Q-Routing algorithms, the Q-Value is stored in a packet traversing the network. Even solutions proposed in these works can achieve low latency, high throughput, and adaptive routing, it can be inefficient when it comes to the Q-Table storage space and the time to reach the best policy.

DRL addresses this challenge by taking advantage of Deep Neural Network (DNN) to train the learning process of reinforcement learning algorithms. Most relevant works are described in

[165][128], where the DRL agent interacts with the network through three signals: state (TM), action (link-weight vector), and reward (improving network performances). DQN [106][107] is one of the most used DRL methods in network routing optimization. Authors in [143] presented an adaptive Deep Q-Network based energy and latency-aware routing protocol (DQELR) that adopts a DQN algorithm with both off-policy and on-policy methods to make routing decisions adaptively in different network conditions. In [96], authors presented DRL-R (DRL-based Routing) to cope with the problem of coexistence of Elephant flow/Mice flow/Coflow and multiple resources (bandwidth, cache and computing). The proposed approach built using DQN and Deep Deterministic Policy Gradient (DDPG). Experimental results demonstrated the effectiveness of the proposed solution in improving network performances. However, these works did not take into account the traffic prediction, since the DRL agent is triggered only once the first *PacketIn* comes to the controller. Moreover, actions depend only on link weights modification not on flow rules installation.

To sum up, these works present the following shortcomings: (i) the non-consideration of the mapping history between the network state and the corresponding action when predicting congestion, and (ii) the non-consideration of current and predicted congestion when rewarding actions. Based on these observations, we propose to train a DQN agent capable of optimizing routing by dynamically choosing the optimal path according to a rewarding function, while taking into account latency, throughput, and packet loss. In addition, we propose to deploy a traffic Prediction module to predict network congestion.

Table 3.1 highlights the added-values of our first contribution in regard to the state-of-the-art approaches, where Rate Monit and Latency Monit denote respectively the monitoring of throughput and latency.

3.2.2 Online based learning for predictive end-to-end network slicing in 5G networks

Recently, there has been a dense specific research interest for network slicing and its impact on resource management. Recall that, network slicing can be applicable in both RAN referred to as radio slicing and transport and CN domains, referred to as transport network slicing [133].

Most of the literature on CN converge toward a common commitment, which consists in separating control plane and user plane functions of the EPC elements (i.e., MME, and S/PGW). In [92][137], authors investigate the feasibilities to virtualize gateway entities used in the CN (SGW, PGW), by proposing new architectures in which they decouple the control plane from the user plane by using SDN controllers and GTP based OpenFlow switches to encapsulate/decapsulate UE data. Although using the SDN controller in the CN gives more flexibility and does not introduce a consequent delay, the authors do not consider the forwarding of the GTP packets in the S1-U interface, where all the UEs traffic flows through one single path. In [112], authors introduce the emerging concept of network slicing and summarizing the preliminary research efforts to enable E2E network slicing of 5G networks. Authors in [132], investigate the problem of mapping core slices for content delivery to a virtualized EPC-based 5G. Their objective is to allocate resources at minimum cost, while meeting

Table 3.1 Added-value of our first contribution

Reference	**Emulated Platform**	**Rate Monit**	**Latency Monit**	**Pred-iction**	**DRL**	**Optimal**	**Heuristic**
[Chowdhury et al. 2014]	✓	✓	✗	✗	✗	✗	✓
[van Adrichem et al. 2014]	✓	✓	✗	✗	✗	✗	✗
[Ming-Hung Wang et al. 2016]	✓	✓	✗	✗	✗	✗	✗
[Yu et al. 2015]	✓	✗	✓	✗	✗	✗	✗
[Reza Parsaei et al. 2017]	✗	✗	✗	✗	✗	✓	✓
[Han et al. 2014]	✗	✗	✗	✗	✗	✓	✓
[Cortez et al. 2006]	✗	✗	✗	✓	✗	✗	✗
[Barabas et al. 2011]	✗	✗	✗	✓	✗	✗	✗
[Azzouni and Pujolle 2017]	✓	✗	✗	✓	✗	✗	✗
[Khodayari et al. 2005]	✗	✗	✗	✗	✓	✗	✗
[Yu et al. 2018]	✓	✗	✗	✗	✓	✗	✗
[Pham Tran Anh et al. 2019]	✓	✗	✗	✗	✓	✓	✓
[Mnih et al. 2013]	✗	✗	✗	✗	✓	✓	✓
[Mnih et al. 2015]	✗	✗	✗	✗	✓	✓	✓
[Su et al. 2019]	✗	✗	✗	✗	✓	✓	✓
[Liu 2019]	✗	✗	✗	✗	✓	✓	✗
Our 1st contribution	✓	✓	✓	✓	✓	✓	✓

QoS requirements over each slice. In [66], authors present a testbed implementation towards E2E slicing of 5G cellular systems, in which, two VNs with different QoS requirements are considered where each network slice tries to satisfy its users requirements through a pre-agreed share of the available resources.

The concept of radio slicing can be traced back to the idea of network sharing defined by 3GPP [12]. Several network sharing models are proposed such as Multi-Operator Core Network (MOCN) where the RAN spectrum is shared among multiple operators [80]. Authors in [69] present RadioVisor, a slicing plane that dynamically slice Three Dimensions (3D) resource grid based on traffic among operators, where the creation and configuration of slices are dynamic based on a logically centralized controller. In [91], authors propose a fully programmable Network Slicing architecture based on the 3GPP Dedicated CN (DCN) and a Flexible RAN to enforce Network Slicing, which enables a two-level MAC scheduler to abstract and share the physical resources among Slices. Authors in [64], present Orion, a RAN slicing system that enables the dynamic on-the-fly virtualization of base stations, the flexible customization of slices to meet their respective service needs and which can be used in an E2E network slicing setting. Moreover, they present a concrete prototype implementation of Orion for LTE, with experimental results. Authors in [166], present a radio slicing architecture based on resource allocation and mobility management for the three major 5G services (eMBB, uRLLC, mMTC). Authors in [47][48] proposed a radio slicing solution for enabling efficient coexistence of eMBB and IoT services, by implementing a northbound SDN application which enables the creation and configuration of IoT slices on demand, while considering the QoS requirements.

Other research works consider the network slicing as a VN mapping/embedding problem, which consists in mapping the VN request describing the set of resources required by a tenant, to the available resources [133]. Authors in [133], present a solution for the dynamic slicing problem in terms of both mixed integer linear programming formulations and heuristic algorithms, where a VN is assigned a resource slice with just enough resources to match the current service needs. However, this work does not consider the traffic load and network congestion and focus only on the resources allocated to VNs.

On the other hand, ML techniques have been used widely to solve various complex problems arising in routing optimization and TE [31]. Much researches leveraging ML in QoS-aware routing problem specifically in SDN based networks, especially when there are multiple flows requesting resources and coexisting in the same network. Authors in [24], showed the out-performance of the ML techniques ARMA/ARIMA on RNN in terms of network routing optimization and congestion prediction based on SDN. LSTM is one of the most successfully RNNs solutions used for handwriting recognition and language modeling [22]. Authors in [114] [159] investigates the prediction accuracy of ML techniques such as Multi-Layer Perceptron (MLP), Multi-Layer Perceptron with Weight Decay (MLPWD), and Support Vector Machine (SVM) in order to maximize the resource usage for network operators. In [129], authors predict wireless traffic based on certain spatial-temporal information by exploiting the RNN. Authors in [161] predict mobile traffic and they compared their approach to random-based predictions.

Table 3.2 Added-value of our second contribution

Reference	**5G Platform**	**Monitoring**	**Radio Slicing**	**Prediction**	**Transport Slicing**	**Core Slicing**	**Heuristic**
[Ksentini et al. 2016]	✗	✗	✗	✗	✗	✓	✓
[Sama et al. 2014]	✗	✗	✗	✗	✗	✓	✗
[Nakao et al. 2017]	✓	✗	✓	✗	✓	✗	✗
[Rayani et al. 2018]	✓	✗	✗	✗	✗	✓	✗
[Gebremariam et al. 2017]	✓	✗	✓	✗	✗	✓	✗
[Katsalis et al. 2017]	✗	✗	✓	✗	✗	✓	✗
[Zhang et al. 2017]	✗	✗	✓	✗	✗	✗	✗
[Costanzo et al. 2018]	✓	✓	✓	✗	✗	✗	✓
[Gudipati et al. 2014]	✗	✗	✓	✗	✗	✗	✓
[Ksentini et al. 2017]	✓	✓	✓	✗	✗	✗	✗
[Foukas et al. 2017]	✓	✗	✓	✗	✗	✗	✗
[Qiu et al. 2018]	✗	✓	✗	✓	✗	✗	✓
[Xu et al. 2019]	✗	✓	✗	✓	✗	✗	✓
[Nikravesh et al. 2016]	✗	✓	✗	✓	✗	✗	✗
[Yamada et al. 2018]	✗	✓	✗	✓	✗	✗	✗
[Raza et al. 2018]	✗	✓	✗	✗	✓	✗	✓
Our 2nd contribution	✓	✓	✓	✓	✓	✓	✓

To sum up, all the above works present the following shortcomings: 1) the non consideration of routing the UE packets in S1-U interface based on GTP, 2) the non consideration of slicing continuity based on the UE connection and finally 3) the non consideration of traffic prediction when it comes to E2E network slicing, and 4) the focusing is principally on the process of instantiating and deploying the Network Slices, while ignoring how they are enforced in the mobile network. Based on these observations, we address the design and implementation of a system capable of: i) forwarding the UE traffic in multi paths in the S1-U interface, by patching OVS to be able to handle GTP traffic, and ii) creating an E2E network slices by considering the bandwidth prediction of radio slices to scale in and out the PN slices, as well as network congestion inside each PN slice, while taking into account the flows QoS.

Table 3.2 highlights the added-values of our second contribution in regard to the state-of-the-art approaches.

3.2.3 Dynamic Clustering of SDN Switches and Controller placement using Deep Reinforcement Learning

In recent years, with the development of distributed controller architecture such as DevoFlow [49], Kandoo [71], HyperFlow [150], Onix [87], the problem of determining the number of controllers and their placements in SDN has received considerable attention and attracted many researchers ([111], [76], [29]).

The authors in [51] present a comprehensive survey on the controller placement problem (CPP) in SDN. The objective of this survey is to introduce the CPP in SDN and highlight its significance. Then, presenting the classical CPP formulation along with its supporting system model as well as discussing a wide range of the CPP modeling choices and associated metrics. In [154], authors tackled the multi-controller placement issue in SDN and proposed a new approach with network partition technique. In this approach, the entire network is divided into multiple sub-networks and for each sub-network, one or more controllers are deployed correspondingly. Specifically, the clustering algorithm is leveraged to partition the network into sub-networks and an optimized K-means algorithm is proposed to shorten the maximum latency between the centroid and associated switches in the sub-network. The authors optimized the K-means algorithm for clustering to minimize the overall latency of the network.

Lange et al. [94] consider node-to-controller latency for their controller placement optimization. They present POCO, a framework for Pareto-based Optimal Controller placement that provides operators with Pareto optimal placements with respect to different performance metrics. This framework does not segment the network into multiple domains by treating the network as a whole and the controllers work collaboratively, which requires frequent exchange of state information between the controllers to achieve an accurate global state.

Authors in [163], defined a capacitated controller placement problem (CCPP), taking into consideration the load of controllers, and they introduced an efficient algorithm to solve the problem. The

objective is to reduce the number of controllers and their loads. However, the placement is not based on these criteria. Instead of considering the propagation delay between the switches and the controller and ignoring the critical factor in the actual network which is the switch weights, the authors in [147] defines a new metric : $total\text{-}flow\text{-}request\text{-}cost$. This metric takes into account the switch weights, switch-to-controller routing costs, and inter controller routing costs, where the goal of this metric is to minimize *Packet-in* messages cost from switch to controller, and information exchange cost between controllers.

In [75], authors first, consider the controller placement problem from the perspective of energy consumption then, they formulate the energy-aware controller placement problem based on a binary integer program (BIP), which has the latency of paths and the load of controllers as constraints for minimizing the energy consumption and they proposed a genetic algorithm to solve the formulated problem. Authors in [100] investigate different approaches to determine the optimal number of controllers for deployment in a given SDN network, by taking into account the latency objective. This study was followed by determining the optimal placements of the SDN controllers. In [146], [43], authors considered several parameters such as the number of controllers, the location of controllers and the set of switches of the network to achieve a high reliability. In [43], authors introduced QoS-Guaranteed Controller Placement problem, which is to place the minimum number of controllers in the network such that response time of controllers can meet a given delay bound. Three heuristics was proposed for the proposed approach: incremental greedy algorithm, primal-dual-based algorithm and network partition-based algorithm. Results show superiority of the proposed incremental greedy method on the other two methods on all input topologies.

Recently, ML techniques have been used widely to solve complicated decision-making complex problems arising in Virtual Network Function (VNF) placement and TE in SDN based networks. Authors in [130] addressed the allocation of Virtual Network Function-Forwarding Graph (VNF-FGs) for realizing network services. First, they modeled the VNF-FG allocation problem as a MDP then they solved it by a DRL approach. The simulation results clearly show the effectiveness of the deep learning process, where the performance of the proposed approach is improved over time. In [108], authors proposed a simple heuristic algorithm to identify number of controllers and their locations in SDN networks leveraging a learning automaton (LA) approach, while ensuring that propagation latency from any node to its closest controller does not exceed a threshold.

Different from these works, we propose a Deep Q-Network based Dynamic Clustering and Placement (DDCP) approach for SDN networks, by exploring the potential of reinforcement learning techniques for the clustering and placement of controllers, while taking into account the propagation delay between the switches and the controller, the controller resource utilization, the intra-cluster latency and the dynamicity of the network.

Table 3.3 highlights the added-values of our third contribution in regard to the state-of-the-art approaches, where Ctrl Plane Load, Ctrl Plane Delay denote respectively the control plane load and latency. Ctrl to SW Load and Ctrl to SW Delay denote respectively the controller to switch load and latency.

Table 3.3 Added-value of our third contribution

Reference	**Ctrl Plane Load**	**Ctrl Plane Delay**	**Data Plane Load**	**Data Plane Delay**	**Ctrl to SW Load**	**Ctrl to SW Delay**	**RL/DRL**
[Müller et al. 2014]	✓	✗	✓	✗	✓	✗	✗
[Wang et al. 2016]	✗	✗	✗	✓	✗	✓	✗
[Lange et al. 2015]	✗	✓	✗	✗	✓	✓	✗
[Yao et al. 2014]	✗	✗	✓	✓	✓	✓	✗
[Tao et al. 2018]	✓	✓	✓	✓	✓	✓	✗
[Hu et al. 2017]	✗	✗	✗	✓	✗	✓	✗
[Mamushiane et al. 2018]	✗	✗	✗	✓	✗	✓	✗
[Tanha et al. 2016]	✓	✗	✓	✓	✗	✓	✗
[Cheng et al. 2015]	✓	✗	✗	✓	✓	✓	✗
[Quang et al. 2019]	✗	✗	✗	✗	✗	✗	✓
[Mostafaei et al. 2018]	✗	✗	✗	✓	✗	✓	✓
[Chen et al. 2018]	✓	✗	✓	✗	✗	✓	✗
Our 3rd contribution	✓	✓	✓	✓	✓	✓	✓

3.3 Conclusion

This chapter provided an overview of the state of the art of the research axes that we identified in this thesis, which are traffic optimization, network slicing and SDN controllers placement. We first discussed different network traffic monitoring approaches. Then, we reviewed ML based QoS-aware routing strategies. Next, we discussed E2E Network slicing approaches based on 5G mobile network architecture and IoT Platform. Finally, we reviewed the SDN controller placement approaches. Moreover, we highlighted our added values by dressing three tables comparing our three contributions with respect to the state-of-the art approaches.

The next chapter will target the first contribution namely, Deep Q-Network and Traffic Prediction based Routing Optimization in Software Defined Networks (DTPRO).

Chapter 4

Deep Q-Network and Traffic Prediction based Routing Optimization in Software Defined Networking

Contents

4.1 Introduction

As stated previously, SDN is an emerging networking architecture, that decouples the network intelligence from the network devices enabling thus the centralization of network intelligence, a flexibility in traffic control and simplicity in network management and operation. However, as the size of the network and the number of flows increase, the computational complexity of the control plane increases exponentially. Therefore, adapting traditional network policies to the continually changing network behavior is challenging. In this way, to effectively avoid congestions and overloading links, both latency and throughput must be monitored continually and proactively in order to quickly route packets to less used links. One of the first SDN protocol standards is OpenFlow [101] that enables direct interaction with the forwarding plane of network devices. Although OpenFlow provides a mechanism to request throughput statistics, latest specifications of this protocol do not provide mechanisms to measure latency.

Moreover, existing routing algorithms are not suitable for SDN due to their limits in convergence, the adaptability to network topology changing, and the absence of a future vision on the evolution of network traffic.

To this end, we propose, in this chapter, a dynamic and efficient TE scheme, called DTPRO, based on a DQN agent and a traffic prediction module in order to optimize the network flow routing. Specifically, our proposed solution consists of three main phases. Firstly, we dynamically optimize the flow routing in the network by training a DQN agent. Secondly, we predict congestion and adjust the DQN reward function to provide better routing configurations. Finally, we route the network traffic based on a set of link weights given by the trained DQN agent, and at the same time reroute the existing ones away from the congested paths by resolving a LP. The formulated LP represents the flow rules placement problem, where the objective is to minimize the total network delay, packet loss and link utilization. Note that, during the third phase, we have proposed an heuristic that interacts with the DQN agent and the traffic prediction in order to solve the formulated LP and optimize network performances. Experimental results using the ONOS controller and Mininet show that the proposed approach provides a promising enhancement against traditional routing algorithms.

The main contributions of this chapter can be summarized as follows:

- First, we train a DQN agent with appropriate states and actions, in order to optimize the flow routing in the network.
- Second, we predict congestion and adjust the DQN reward function to provide better routing configurations.
- Third, we mathematically model the QoS-aware routing problem as a LP, which takes as inputs routing strategy given by the trained DQN agent and the predicted traffic. Our objective is to minimize the E2E delay, E2E link utilization and E2E packet loss. Then, we propose a simple yet efficient heuristic algorithm called DTPRO to solve the LP.

- Fourth, we implement the DTPRO approach using ONOS controller and Mininet.

The remainder of this chapter is organized as follows. In Section 4.2, we discuss the architecture of the proposed framework and the rules placement algorithm. Section 4.3 evaluates the proposed method. We finally conclude this chapter in Section 6.5.

4.2 Proposed DQN and Traffic Prediction based Routing Optimization (DTPRO) Approach

Combining ML techniques with SDN is crucial to improve network performances. In this section, we first present our DTPRO approach. We start by explaining the global architecture. Thereafter, we describe the Network Measurement modules (i.e., Latency Measurement, Statistics), the DQN and Traffic Prediction modules. Finally, the mathematical model and the proposed heuristic will be described in the Proactive Forwarding module.

4.2.1 DTPRO Architecture

Although SDN provides the centralization of network intelligence, a flexibility in traffic control and simplicity in network management and operation is still needed. KDN has been introduced as a new paradigm to bring the intelligence to the network management, using the telemetry data, which suggests to add the KP to the conventional SDN paradigm, by adopting Artificial Intelligence (AI)and cognitive system to build the network model.

In this context, we propose to design our framework according to the KDN paradigm, in which we exploit the control plane to have a global view of the network (cf. Fig. 4.1). As depicted in Fig. 4.1, our proposed architecture consists of four planes: Data plane, Control plane, Management plane and Knowledge plane.

The Data plane consists of programmable forwarding devices in charge of data packet processing and forwarding. These devices have no embedded intelligence to take decisions and rely on the control plane to populate their forwarding tables and update their configurations based on the OpenFlow protocol.

The Control plane is considered as the brain of the SDN network, which incorporates the whole intelligence by centralizing the management and global view of the network in a specialized central controller, in which we deploy two main modules: Network Measurement and Proactive Forwarding modules. The Network Measurement module consists of two sub-modules: Statistics which continuously collects metrics such as the number of packets and bytes per flow to measure the throughput, and Latency Measurement. This latter sub-module measures continuously the network latency by periodically sending a packet probe to the data plane. On the other hand, the Proactive Forwarding module is responsible for determining the optimal routing strategy as well as the predicted traffic from the KP by resolving an optimization problem as will be explained later in sub-section 4.2.5.

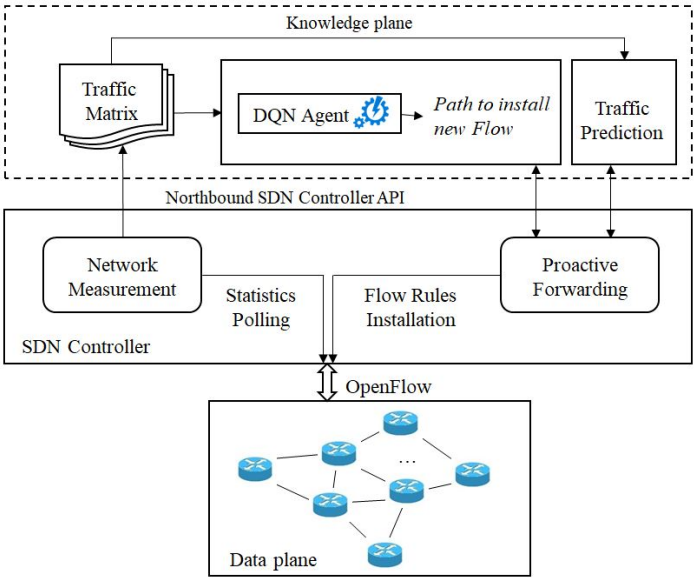

Fig. 4.1 Deep Reinforcement Learning and Traffic Prediction based Routing Optimization Architecture

The Management plane ensures the correct operation and performance of the network by collecting the network measurement from the control plane, specifically from the Network Measurement module, in order to provide network analytic. The collected statistics will be analyzed and sent to the KP.

Finally, the KP exploits the control and the management planes by taking the data from the Management plane as input to be fed to ML algorithms, which convert them to the form of knowledge. Precisely, they learn the behavior of the network, by processing the collected statistics, then extract the optimal paths, representing the knowledge, to route flows by deploying a DQN agent, and finally, predict network congestion using prediction methods (i.e., LSTM, ARIMA, LR) in the Traffic Prediction module. Note that the routing strategy is determined by the DQN agent by exploiting the historical data of routing configurations.

In what follows, we detail further these modules.

4.2.2 Network Measurement

The Network Measurement module ensures the data plane monitoring based on the OpenFlow protocol, which is crucial in network management, and helps Operators to make decisions about Load Balancing, Routing, QoS, SLA and so on. The collection of statistics from the data plane can be either active or passive. In the active mode, the Controller sends and receives probe packets to the entire data plane network to measure statistics such as the Round-Trip-Time (RTT), Latency, Packet Loss. On the other hand, the passive mode corresponds to querying the statistics information from switches by using standard OpenFlow messages.

As stated earlier, the Network Measurement module consists of two sub-modules: Statistics and Latency Measurement. The Statistics module monitors the data plane according to the passive mode

using the OpenFlow standard messages to read the information from the control plane. Specifically, it uses the `OFPStatsReply` message type, in which, the switch periodically reports its statistics through two types of messages: `PortStatsReply` and `FlowStatsReply`, to respectively measure the throughput and the per-flow size. Note that, the throughput can be measured based on the number of sent/received packet or bit reported in the `PortStatsReply` message.

The Latency Measurement module, on the other hand, uses the active monitoring mode by sending periodically a packet probe to the data plane to measure the latency based on the notifications' arrival times at the control plane. Specifically, it consists in firstly, measuring the time that takes a packet probe from the controller and traverse the path and return back to the controller ($Total_{Delay}$). Secondly, it measures the time that takes the packet probe to go from the controller to the first switch (i.e., t_1) and the last switch on the path (i.e., t_2), as shown in Fig. 4.2. [145][24].

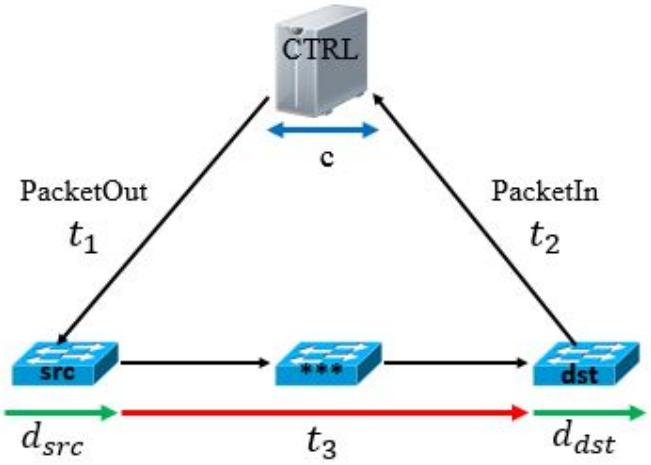

Fig. 4.2 Latency computation mechanism

Some specific monitoring rules must be installed at the first and last switches (*src*, *dst*) respectively on the path that contains "`Send to Controller`", in order to measure the delay between the controller and the switches (i.e., t_1, t_2, as shown in Fig. 4.2) and to send the packet probe back to the controller at the last switch of the path. The switches between the first and the last switches in the path forward the packet probe by a pre-installed forwarding flow rules corresponding to the packet probe. The delay needed corresponds to the time t_3, and can be determined as follows:

$$t_3 = Total_{Delay} - (t_1 + t_2 + d_{src} + d_{dst} + c) \tag{4.1}$$

Where d_{src}, d_{dst} are the processing times in the devices (*src*, *dst*) respectively, and c is the processing time in the controller. In order to accurately estimate the $Total_{Delay}$, t_1 and t_2 delays, the time is encapsulated in nanoseconds in the packet probe. Moreover, the monitoring rules installed to guide the packet probe do not interfere with the normal traffic.

It is worth noting that the Latency Measurement module is designed according to the OSGi standard that implements an application as a complete and dynamic component model. It is indeed composed of four components: Monitoring Flow Rules Installation, Probe Packet Generator, Packet Processing and Latency Measurement, as shown in Fig. 4.3. When activating the Latency Measurement module we start by installing the probe packet flow rules to guide the probe packet along the path to be monitored. The Probe Packet Generator will then send periodically a specific probe packet to the data plane, that matches the monitoring flow rules already installed by the previous component. Thereafter, the Packet Processing component listens to the incoming `PacketIn`, then, if the latter corresponds to the probe packet, the Packet Processing component extracts the times between the controller and switches and sends them to the Latency Measurement component that measures the latency as explained above and stores the latency in a centralized database.

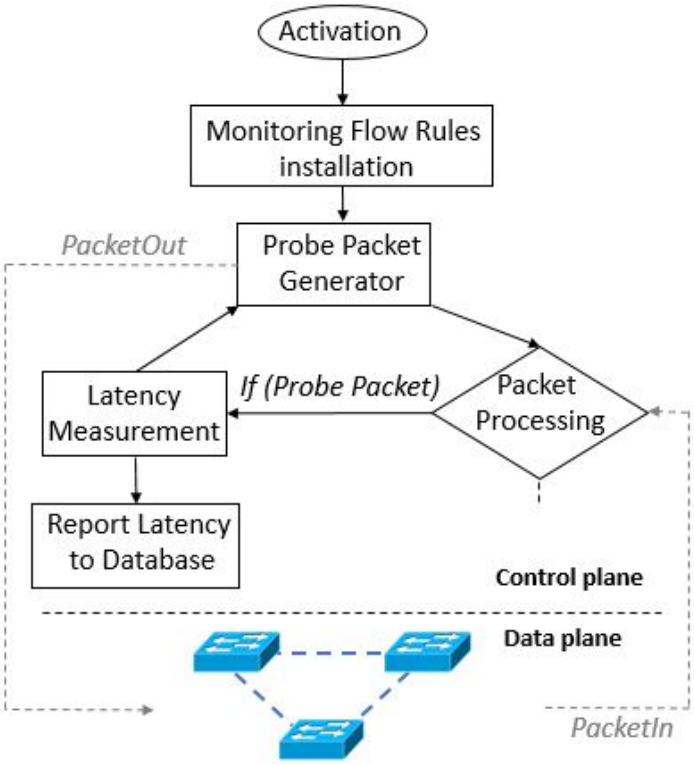

Fig. 4.3 Latency Measurement Design

4.2.3 Traffic Prediction Models

In order to avoid congestion and improve network performances, it is important to predict the future evolution of network traffic.

To do so, we propose here to use the well-known LSTM model to predict the network latency and compare it with two other prediction models: ARIMA, and LR. In what follows, we detail these three models and show how we adapt them to predict the E2E network latency.

Linear Regression (LR)

as described in figure 4.4, a valuable way of modeling the relationship between the measuring time of delays (i.e., the explanatory variable t) and estimated ones (i.e., the dependent variable $\widehat{Delay}$) is by using LR as indicated in equation (2.5). In this way, we can build the regression equation between the time of sampling and the delay, by using the equation (4.2) :

$$\widehat{Delay} = \lambda.t + \mu + \varepsilon \tag{4.2}$$

Where t is the sampling time, λ and μ are obtained by applying the OLS method on the set of measurements: (i.e., Delay, Measurement time t) so that the gap between the sum of the squares of the differences between the predicted variables (i.e., $\widehat{Delay}$) and the explanatory variables (i.e., t) is minimized. Note that the regression quality is determined by using the correlation coefficient [82].

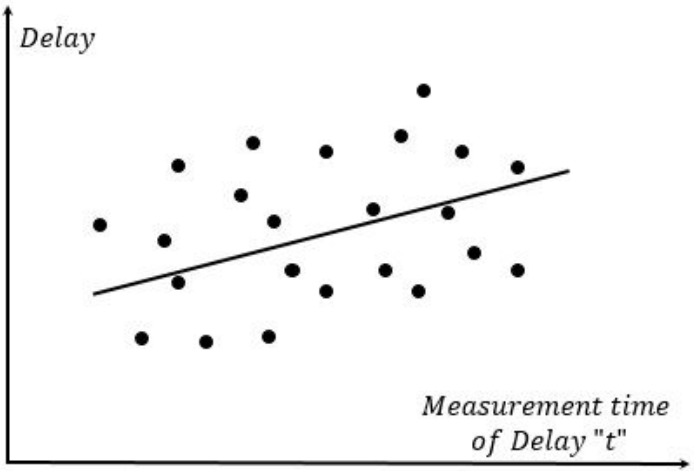

Fig. 4.4 Linear Regression LR

ARIMA Model

we propose to model the time series latency according to the ARIMA(p,d,q) model [37]. In which the latency is considered as a stochastic process, that can be explained as a linear combination of p past observations (i.e., Latency Measurement): $D_{t-1}, D_{t-2}, ..., D_{t-p}$, and q past white noises: $e_{t-1}, e_{t-2}, ..., e_{t-q}$, together with a random error in the same series, as shown in the following equation:

$$D_t = c + \phi_1 D_{t-1} + ... + \phi_p D_{t-p} + e_t + \theta_1 e_{t-1} + ... + \theta_q e_{t-q} \tag{4.3}$$

Where, p, the number of past observations that D_t depends on, which represents the Auto-Regressive (AR) degree, q represents the number of Moving Average (MA) order and d is the degree

of differentiation [37]. It is worth noting that D_t is the latency to be predicted using previous samples of the latency time series.

Recall that the identification of the ARIMA model involves several steps where the mains are: i) checking the stationarity to be able to use correctly the ARIMA model, ii) order (p,d,q) must be estimated, we refer to Autocorrelation Function (ACF) and PACF to estimate these orders [37].

Long Short-Term Memory (LSTM)

with the continuous development of internet technology, network services, the network traffic is constantly expanding showing more burst and self-similar, and the complexity is becoming higher and higher. Consequently, the traffic evolution shows non-linearity. This makes the RNNs LSTM well placed to learn complex non-linear patterns. One of the advantages of LSTM is that several models can be used based on each type of time series forecasting such as the shape of the input/output of the NN. In this way, we propose to model the traffic latency prediction using LSTM model [74] for multi-step time series forecasting as follows:

- LSTM Inputs:
  - Sample: S
  - Time Steps: T
  - Features: n
  - Learning Window: w
  - Units: nbr_units
  - Delay Vector: $\widehat{D} = \{\widehat{d}_{(u,v)}\}$
- LSTM Output:
  - Predicted Delay Vector: $\widehat{D} = \{\widehat{d}_{(u,v)}\}$

Where the samples S represents the number of training examples, the Timesteps T refers to the LSTM memory capacity and the Feature n is the amount of features in every time step. These three elements construct the three-dimensional structure $[Samples, Timesteps, Features]$ expected by the LSTM model. The Learning Window w refers to the number of previous time-slots to learn from in order to predict the future Delay vector. This parameter is used to avoid learning in long sequences that can result in high computational complexity. The nbr_unit parameter represents the number of neurons and finally the Delay vector refers to the previous latency Measurements to be fed to the LSTM Model. The output of the LSTM model is the predicted vector of link delays. Figure 4.5 shows the overall architecture of the used LSTM model.

It is worth noting that, before applying an LSTM model on the dataset and predicting, we need to transform the data as follows: i) the Time series data must be stationary, ii) the Time series must be

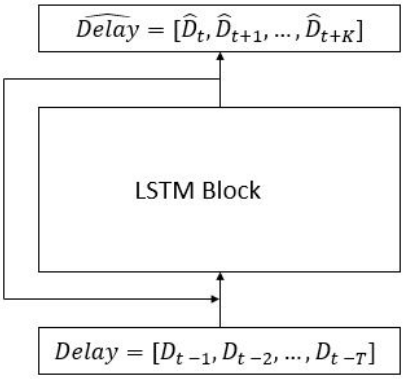

Fig. 4.5 LSTM based Latency prediction

transformed into a supervised learning problem. More specifically, the previous observations are used to predict the current observation, and iii) It was necessary to have a specific scale responding to the default tangent hyperbolic activation function *Tanh* of the LSTM model.

4.2.4 Routing Optimization Model based on DQN

To meet the requested QoS, flows must be routed following the best routing strategy. As the traffic state (stream) depends on the type of data transported in the network, choosing the best routing path to that stream improves its QoS. To this end, we propose here to deploy a DQN agent that dynamically determines the optimal path. We model this DQN as follows:
Given a network topology represented as a non-oriented graph $G(V,E,C)$ where V, E and C are, respectively, the vertex, the edge and the link capacity sets, and $|V| = n$ represents the number of network nodes, the DQN agent interacts with the environment through three signals: State, Action, and Reward. "State" is the $(n \times n)$ Traffic Matrix representing the current network load. The "Action" taken by the agent is the link-weight vector, and the "Reward" r of the agent is related to the QoS parameters, which are mainly: the average of Network latency ($\overline{L}$), the average of Data rate ($\overline{W}$) and the average of Packet Loss ($\overline{PL}$). In this case, the Reward r can be determined as follows:

$$r = \alpha.\overline{W} - \beta.\overline{L} - \gamma.\overline{PL} \tag{4.4}$$

Where $\alpha, \beta, \gamma \in [0,1]$ are the adjustable weights determined by the routing strategy. Our objective here is to determine the optimal policy π mapping the set of states to the set of actions in order to maximize the reward r. Note that the routing strategy is determined by a set of weights and periodically updated at the beginning of each time epoch T_{DQN}, which is initialized to 1 hour in this work.

4.2.5 Proactive Forwarding Formulation and Resolution

The Proactive Forwarding module is responsible for routing flows according to the optimal routing strategy, by using the current and predicted Traffic Matrix. It is designed according to the OSGi model and combines four components: (i) Load Predicted Traffic, (ii) Load Weights, (iii) Packet Processing and (iv) Flow Rules Generator, as shown in Fig. 4.6. The two sub-modules Load Predicted Traffic and Load Weights are respectively responsible for collecting the predicted traffic and the set of link weights from the KP layer. The Packet Processing listens to the `PacketIn` coming from the data plane and finally the Flow Rules Generator is responsible for flow routing. Two events automatically trigger the latter: the first one corresponds to generating flow rules for new incoming flows by using the collected weights by Load Weights sub-modules, to calculate their corresponding paths, and the second one happens when a congestion is detected, where the action is to reroute the current traffic to the less used paths.

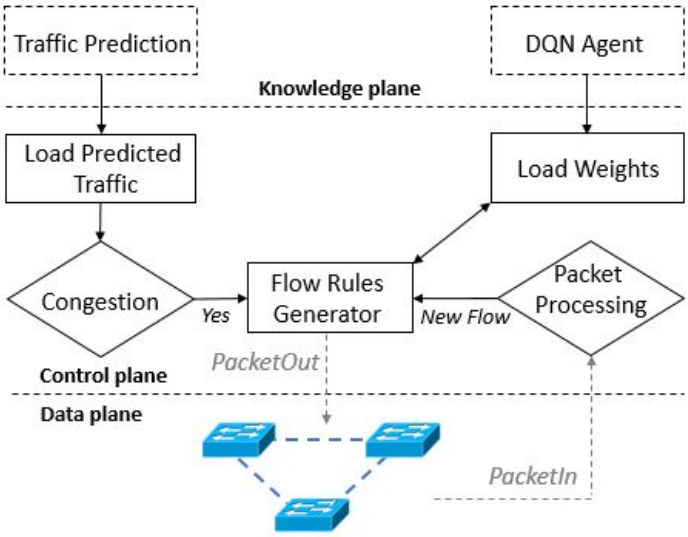

Fig. 4.6 Proactive Forwarding Design

The idea here is to give more accuracy to the routing strategy selected by the DQN agent as explained in the previous sub-section 4.2.4, by incorporating the prediction aspect, where the objective is to determine which link to route which flow in order to minimize the E2E network latency and balance the network load by minimizing link utilization.

In this context, the physical infrastructure can be represented by a capacitated graph $G(V,E,C)$, where V denotes the set of nodes or switches and E the set of edges representing the physical or virtual links in the network. Each link is characterized by its current and predicted propagation delay $d_{(u,v)}, \widehat{d_{(u,v)}}$, bandwidth capacity $C_{(u,v)}$ and threshold $TS_{(u,v)}$. We assume the network has m nodes ($|V| = m$), ne links ($|E| = ne$) and l flows. The current and predicted Traffic Matrix $TM, \widehat{TM}$ representing, respectively, the current and predicted volume of traffic flows between all pairs of origin

and destination nodes in the network. Each flow f_i is characterized by its size $FS_{(f_i,u,v)}$ and the path to which is affected P_{f_i}.

The network may have the capacity limitation constraint. To this end, we define three variables $E_{(f_i,u,v)}$, $LU_{(u,v)}$, $MLU_{(u,v)}$ as follows :

$$E_{(f_i,u,v)} = \begin{cases} 1, & \text{if } f_i \in (u,v) \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

$$LU_{(u,v)} = \sum_{f_i \in F} (E_{(f_i,u,v)} \times FS_{(f_i,u,v)}) \tag{4.6}$$

$$MLU_{(u,v)} = C_{(u,v)} * \lambda_{(u,v)} \tag{4.7}$$

Where the symmetric Binary matrix $E = E_{(f_i,u,v)}$ denotes whether the flow rule f is allocated to the link (u,v) or not. Also $LU_{(u,v)}$ denotes the current link utilization and $MLU_{(u,v)}$ represents the Maximum Link Utilization of link (u,v). In the equation (4.7), λ depends on link characteristics and $C_{(u,v)}$ is the link capacity.

As the network traffic may have several classes based on latency sensitivity, it will be necessary to give more priority to those latency sensitive applications. To this end, we also define two variables: δ and st_{f_i,s_j}. We assume each switch has q priority, the variable δ represents the priority of flow f_i in switch s_j and the decision variable st_{f_i,s_j} denotes whether the flow f_i is being routed or waiting in the queue.

$$\delta = \begin{cases} \delta_{f_i,s_j}, & \delta_{f_i,s_j} \in [1,q], \text{if } f_i \in s_j \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

$$st_{f_i,s_j} = \begin{cases} 1, & \text{if } f_i \text{ is routed} \\ 0, & \text{if } f_i \text{ is waited} \end{cases} \tag{4.9}$$

Hence, our problem can be formulated mathematically as a LP as follows:

- Objective:
  - Minimize $\tau.D + \nu.LU + \zeta.PL$
- Subject to:
  - Delay limitation: $\forall (u,v) \in E$:
    $Max(d_{(u,v)}, \widehat{d}_{(u,v)}) \quad < \quad TS_{(u,v)}$
  - Link capacity limitation:
    $\forall (u,v) \in E : LU_{(u,v)} < MLU_{(u,v)}$

- Path capacity limitation:
  $\forall(u,v) \in FP$: $MLU_{(u,v)} - LU_{(u,v)}$
  $\leq Cap\_Av\_Path(FP)$
- Path Flow priority:
  $\forall s_k \in Path\ P, \forall f_i, f_j \in s_k^2$:
  $\delta_{f_i,P} > \delta_{f_j,P} \Rightarrow \sum_{s_k \in P} st_{f_i,s_k} \geq \sum_{s_k \in P} st_{f_j,s_k}$
- Demand satisfaction:
  $\forall(u,v) \in Path\ P, \forall f_i \in F$:
  $\sum_{(u,v) \in Path, f_i \in F} FS_{(f_i,u,v)} = dem_i$

The delay and link capacity limitation constraints are specified so that we force each link to not be delayed and overloaded. The Path Flow priority constraint ensures that the flow with high priority must be routed first and finally the Demand satisfaction constraint ensures that the traffic demand dem_i sent through any path source src_i must be equal to that in the path destination dst_i. Our objective is to minimize the network delay D, the link utilization LU and the packet loss PL, while the total demands are always satisfied. Note that τ, ν, ζ are the weighting factors related to the degree of importance of D, LU and PL, respectively. These latters are defined as follows:

$$D = \frac{1}{ne} \sum_{(u,v) \in E} d_{(u,v)} \tag{4.10}$$

$$LU = \frac{1}{ne} \sum_{(u,v) \in E} LU_{(u,v)} \tag{4.11}$$

$$PL = \frac{1}{ne} \sum_{f_i \in F} (dem_i - \sum_{(u,v) \in P_{f_i}} FS_{(f_i,u,v)}) \tag{4.12}$$

The formulated problem can be considered as a multi-commodity flow problem, which are known to be NP-Hard. Furthermore, it is assumed to be solved by the SDN Controller for each incoming flow. However, as the size of the network and the number of flows increase, the computational complexity increases exponentially. Clearly, such approach is not feasible in practice, since it generates high overhead due to the frequent updates of the flow tables.

To cope with this problem, and reduce the computation time and complexity, we propose here a simple yet efficient heuristic algorithm, called DTPRO algorithm. DTPRO allows a high-quality traffic allocation, while minimizing the total network latency and packet loss. Algorithm 1 shows the detail of the proposed heuristic. It takes as inputs the current and predicted Traffic Matrix (TM and $\widehat{TM}$, respectively), the current and predicted matrix of link delays (D and $\widehat{D}$, respectively), and the matrix of link delays threshold TS that defines the maximum tolerable delays of each link (u,v). It is worth noting that, the DTPRO is executed for each incoming flow as well as when detecting a congestion.

Algorithm 1: DTPRO algorithm

1: **procedure** $(G(V,E,C),TM,\widehat{TM})$
2: **schedule_every** (T_{DQN})
3: $Links\_Weights \leftarrow get\ optimal\ Weights$
4: $from\ DQN\ agent$
5: $Update\_Net\_Config(Links\_Weights)$
6: **end schedule**
7: $t \leftarrow 0$
8: **while** $(d_{(u,v)} > TS_{(u,v)}$ **or** $\widehat{d_{(u,v)}} > TS_{(u,v)}$ **or** $LU_{(u,v)} > MLU_{(u,v)})$ **do**
9: $CP \leftarrow Get\_Congested\_Path(u,v)$
10: $F \leftarrow Sorted\_Flows\_Priority(CP,\delta)$
11: $R \leftarrow F[t].flow$
12: $SP \leftarrow Sorted\_Backup\_Paths(R)$
13: **for** *each path* $p \in SP$ **do**
14: **if** p *can route* R **then**
15: $Install\_Rule(p,R)$
16: $Remove\_Rule(CP,R)$
17: $Break$
18: **end if**
19: **end for**
20: $t \leftarrow t+1$
21: **end while**
22: $Adjust\_Reward(\alpha,\beta,\gamma)$
23: **end procedure**
24: $\omega \leftarrow 10^{-2},\ \alpha,\beta,\gamma \leftarrow 10^{-1}$
25: **procedure** ADJUST_REWARD(α,β,γ)
26: **if** $d_{(u,v)} > TS_{(u,v)}$ **or** $\widehat{d_{(u,v)}} > TS_{(u,v)}$ **then**
27: $\beta \leftarrow \beta+\omega$
28: **if** $LU_{(u,v)} > MLU_{(u,v)}$ **then**
29: $\alpha \leftarrow \alpha+\omega$
30: $\gamma \leftarrow \gamma+\omega$
31: **end procedure**

Our algorithm works as follows. At the beginning of each time epoch T_{DQN}, it requests the DQN agent to get the optimal link weights *Links_Weights*, then it updates the network configuration (lines 2-6). It is worth noting that, the T_{DQN} corresponds to the time interval to apply a new routing strategy obtained from the DQN agent. Moreover, to show the impact of using different network configurations or routing strategies, this parameter is initialized to 1 hour. However, this parameter can be modified by the network administrator in the order of days or weeks.

Thereafter, since we measure continuously current and predicted Traffic Matrix, if a congestion occurs, in which the current or predicted link delay is greater than a certain threshold or the link is overloaded (line 8), our algorithm finds the flow rule R corresponding to the flow with maximum size in the congested path CP. Then, it sorts all other paths (denoted by SP) by the delay matching the flow

rule R (lines 9-12). In this case, the flow rule R must be rerouted to a path in SP (lines 13-19). If no path in SP can accommodate the corresponding flow size, the flow is discarded and our algorithm goes to the next flow in the CP (line 20). Finally, it adjusts the parameters (α, β, γ) of the DQN reward function r, so that the DQN agent avoids a similar transition (line 22). The *Ad just_Reward* function (lines 25-31) adjusts the parameter β (equation. 4.4) if a link is delayed or predicted to be delayed. On the other hand, the α and γ parameters are adjusted when a link is overloaded or is predicted to be overloaded.

4.3 Performance Evaluation

In this section, we evaluate the efficiency of our proposed approach. We start by presenting our environmental setup. Then, we present the experimental results.

4.3.1 Experimental setup

First, we implement the Network Measurement module (Latency Measurement and Statistics) as well as the Proactive Forwarding module as cooperating modules for the Java-based OpenFlow controller ONOS [26], based on our previous framework developed in [34]. Then, the DQN agent and the Traffic Prediction are implemented based on Python[1], which are dockerized on Docker[2] Containers. Note that the DQN agent and the Traffic Prediction interact with the Proactive Forwarding module based on the ONOS Northbound API. We used the network emulation tool OVS [126] to implement the experimental topology illustrated in Fig. 4.1. To generate traffic among hosts, we used Iperf[3], which consists of a set of flows between a set of hosts (h_1, ..., h_8). The Network Measurement module collects statistics (network latency, throughput, and per-flow size) from the devices and reports those time-series statistics to the InfluxDB[4] database each time interval Ti which is equals to 5 seconds. The link-labels in Fig. 4.7 show the capacity C for each link.

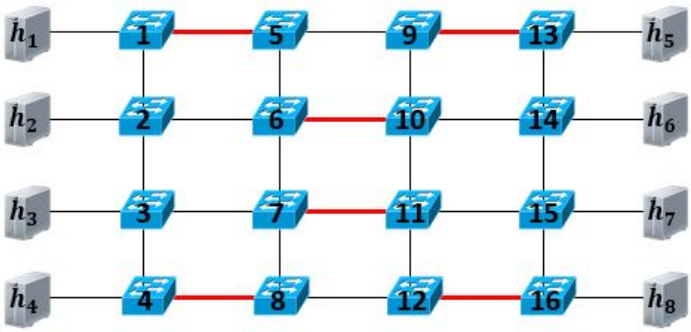

Fig. 4.7 Emulated Topology

[1]https://www.python.org/
[2]https://www.docker.com/
[3]https://iperf.fr/
[4]https://www.influxdata.com/time-series-platform/influxdb/

We built and trained the DQN model by using the Tensorflow[5] library, by deploying separately two NNs, one for *Q-Network* and the other for *Q-target*, which have the same architecture. The parameters of the DQN are illustrated in table I. In particular, both the *Q-Network* and the *Q-target* consists of 2 dense layers. In order to show the impact of the action space size, which corresponds to the number of network configurations for each input traffic matrix state, we trained separately three DQN agents with 24, 60 and 120 network configuration, respectively. The state vector size is 48, which corresponds to the input shape of each NN, and the outputs of the NNs for each training correspond to the actions space sizes 24, 60 and 120, respectively. Consequently, the architectures corresponding to each training are (48, 24), (48, 60) and (48, 120). During the training phase, we adopt ε-greedy method as action selection method and the final exploration rate is fixed at 0.2, while the *Q-target* parameters are copied from the *Q-Network* every 300 steps. The learning rate and discounted factor are, respectively, 0.01 and 0.95. Finally, as we trained three different DQN agents, each training corresponds to 120 episodes.

Table 4.1 DQN parameters for DTPRO

Name	Values
Dense layers	2
State size	48
Action space size / Output shape	24,60,120
Q-target network update frequency	300
Learning rate	0.01
Discounted factor	0.95
Mini-batch size	32
Final exploration rate	0.2
Memory size	500 units
Number of episodes	120
Episode capacity	360000 steps

On the other hand, we built and trained the LSTM model by using Keras[6] Library, where the number of dense layers is 2. We used Adam [86] for learning the NN parameters with a learning rate 0.01. We used *Relu* as activation function. The LSTM method outputs a predicted vector. To this end, we referred to the traffic matrices used for DQN, by dividing them into multiple input/output samples and the size of the output sample corresponds to the prediction interval *Pi*.

Note that, in order to improve the performances, the training of ARIMA, LSTM and DQN models is done offline. These trained models are then saved in such a way that only one step is needed to get the optimal path from the DQN agent or the predicted traffic from LSTM or ARIMA. In parallel, these models continue to learn from these new steps.

[5]https://www.tensorflow.org/
[6]https://keras.io/

4.3.2 Prediction accuracy evaluation

To quantitatively evaluate the overall performance of our prediction models, we use the Root Mean Square Error (RMSE), defined as the difference between the predicted values and the actual values by computing the root of the average sum of squared errors. It can be expressed as follows:

$$\text{RMSE}(model) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\widehat{x_i} - \text{x}_i)^2} \tag{4.13}$$

where $\widehat{x_i}$ and x_i are, respectively, the normalized predicted value and the normalized actual value for the same time interval and N corresponds to the total number of predictions or the size of the prediction interval Pi, which is initialized to 5 seconds.

4.3.3 Experimental results

In order to evaluate the performance of our proposed DTPRO solution, we first select the best predictions models to the network traffic evolution. Then, we evaluate the DQN model, while using different parameters and configurations and finally, we compare the proposed solution with different routing schemes, while taking into account the obtained traffic predictions models.

First, we estimate the best ARIMA model to the proposed network traffic evolution, which is done by estimating the ARIMA parameters [37] : the order of the AR term (p), the order of the MA term (q) and the number of differencing needed to make the time series stationary (d). The time series should be stationary by having values around a defined mean. However, the network traffic could have non-stationary evolution over time. To this end, differencing the time series is a one way to make it stationary and the right order of differencing corresponds to (d). In our experiments, we referred to the ACF plot [37] for differencing and the ADF [110] to check if the series is stationary. In this way, while the differenced series are not stationary, we increment d. The order of the AR term p corresponds to the number of lags ($\text{x}_{t-1}, \text{x}_{t-2}, ..., \text{x}_{t-p}$) that must be used as predictors. In this work, we referred to the PACF [37] to estimate the parameter p, by checking if there is a correlation between a specific lag and the time series. Finally, the parameter q corresponds to the number of lagged prediction errors needed in the ARIMA model and similar to PACF to estimate the parameter p, we used the ACF to estimate the parameter q.

Fig. 4.8 compares the prediction and forecasting accuracy of a set of best ARIMA models to our network traffic, using the RMSE metric. The parameters are estimated based on methods presented above. In this way, the forecasting term is used to indicate the prediction of future values given past values of time series, while the prediction term is used to do estimation whether in future, current or past. From this figure, we can see that models with $d = 1$ give a good prediction and forecasting accuracy regarding the network traffic. The evolution can be repeated over time. We hence refer to this model with best prediction accuracy as the one with ($p = 9, d = 1, q = 1$). This model will be used in our next experiments.

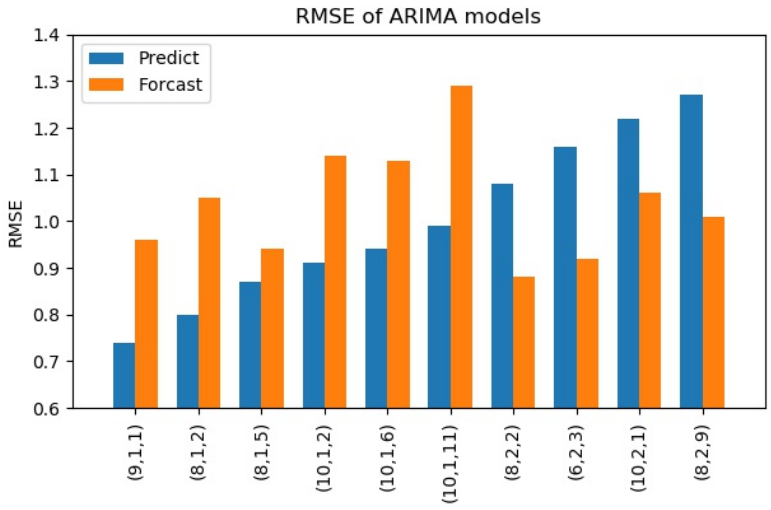

Fig. 4.8 Selecting ARIMA models while varying the p, d, q parameters

Regarding the LSTM model, in order to ensure that its estimation accuracy is good enough and to avoid the over-fitting problem when the network is trained, it is required to find the best number of neurons and the number of training epochs. To this end, we plot in Fig. 4.9 the average of the Loss function and the average of RMSE under different number of training epochs. In this way, we measure both the RMSE and Loss function for each 500 epochs. For the sake of simplicity, we fixed the number of nodes to 100 nodes. In this experiment, we decided about the number of training epochs when the Loss function converges and starts to be stable, which is ensured from this figure by both the RMSE and Loss function. We can see also that both the RMSE and Loss function start to be stable and achieve a good estimation accuracy when the number of training epochs is 9000. The next experiment will focus on identifying the number of hidden nodes.

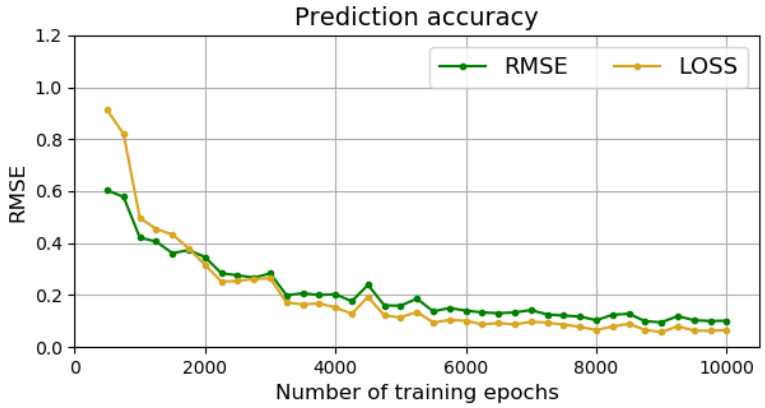

Fig. 4.9 LSTM Loss and RMSE under different number of training epochs

As stated earlier, the number of hidden nodes of the LSTM network is crucial to achieve a stable network configuration. To this end, we plot in Fig. 4.10 the average of the Loss function and the average of RMSE under different number of hidden nodes by increasing the number of hidden nodes by 10 for each measurement. Similar to the previous experiment, we decided about the number of hidden nodes when the Loss function converges and starts to be stable. We can clearly see from that figure that both the Loss function and the RMSE converge and start to be stable and achieve a good estimation accuracy when the number of hidden nodes is 150.

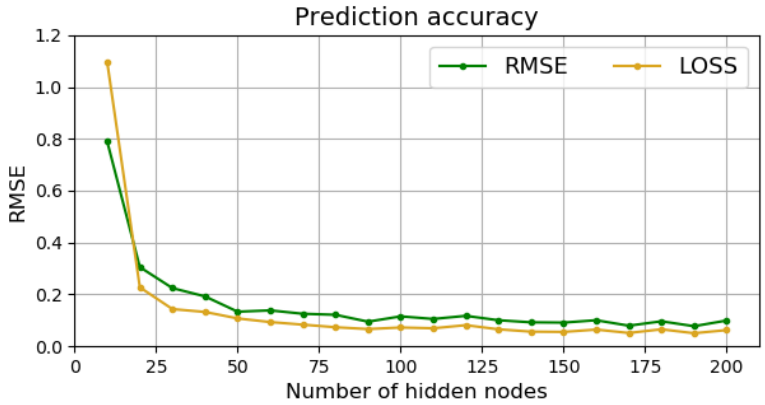

Fig. 4.10 LSTM Loss and RMSE under different number of hidden nodes

From the results obtained in the two previous experiments, we decided about the best LSTM model, which corresponds to 9000 training epochs and 150 hidden nodes, alongside to the initial parameters presented above in this section.

On the other hand, the LR parameters λ and μ, as indicated in equation (2.1), are estimated online, which means that when the prediction is triggered, the Traffic Prediction based LR looks for the N previous measurement, to identify the parameters λ and μ, then it predicts the future evolution of network traffic by using the equation (2.1). In Fig. 4.11, we plot the variation of λ, μ parameters while training the LR prediction model. We can see that these parameters are dynamically changed while changing the data intervals.

Fig. 4.12 compares the prediction accuracy of the different methods presented earlier (i.e., LSTM, ARIMA and LR) by measuring the impact of the prediction interval *Pi* on the RMSE metric, which is increased by 3 seconds for each measurement. Recall that the ARIMA model used in this experiment is $(p = 9, d = 1, q = 1)$ and the used LSTM model is the one with 9000 training steps and 150 hidden nodes and the LR model is dynamically estimated.

From Fig. 4.12, we can see that the RMSE of LSTM is quasi-stable when increasing the prediction interval *Pi*. We can see also that ARIMA achieves a clear stability when increasing the prediction interval *Pi*, due to the nature of the network traffic, which is periodic in a certain level. However, the increase in the LR RMSE is clearly visible, which can be interpreted as the increasing of the distance between the predicted points in the line $\lambda.t + \mu$ and the dispersed traffic points. Furthermore, we can

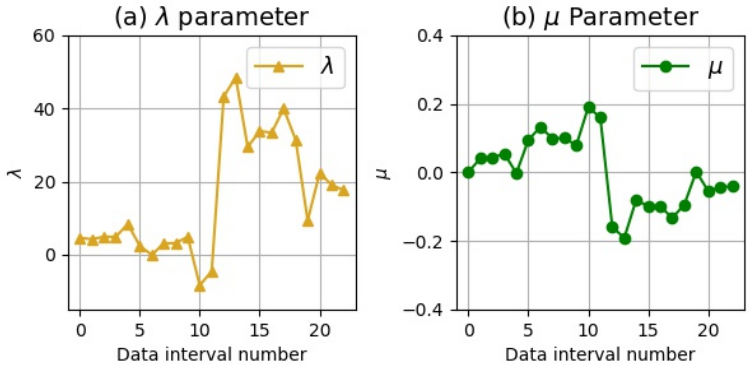

Fig. 4.11 LR parameters λ, μ under different data intervals

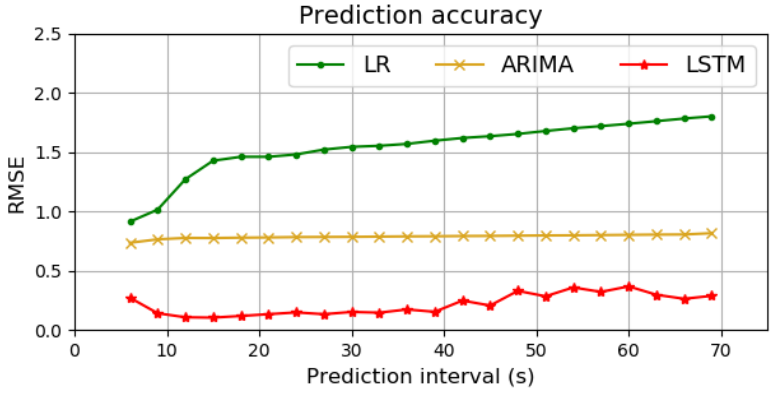

Fig. 4.12 RMSE of LSTM, ARIMA and LR prediction methods

see that ARIMA outperforms LR due to its capacity to estimate the correlation with the previous lags of the time series. Finally, it is clearly visible that LSTM outperforms all others prediction methods due to its capacity to learn long term dependencies. In what follows, we determine the best DQN model for the proposed network traffic evolution.

Recall that, the principal idea of the proposed DQN model is to learn the best policy mapping the set of states (i.e., traffic matrices) to the set of actions (i.e., network configurations), while maximizing a numerical reward defined in equation (4.4). In this way it is required to find the best action structure, in order to ensure a good estimation accuracy of the DQN model, while avoiding the over-fitting problem when the DQN is trained. To this end, we plot in Fig. 4.13 the evolution of the average of the Loss function and the average of the reward function under different number of training episodes and the three action space sizes (24, 60, 120). The DQN is trained based on the initial parameters as stated earlier in this section. In the experiment, we decide about the best action structure when the Loss function converges and the reward function starts to be stable.

From Fig. 4.13 we can see that the Loss function for the three action space configuration converge. However, we can observe from Fig. 4.13 (b) that the more we increase the action space capacity,

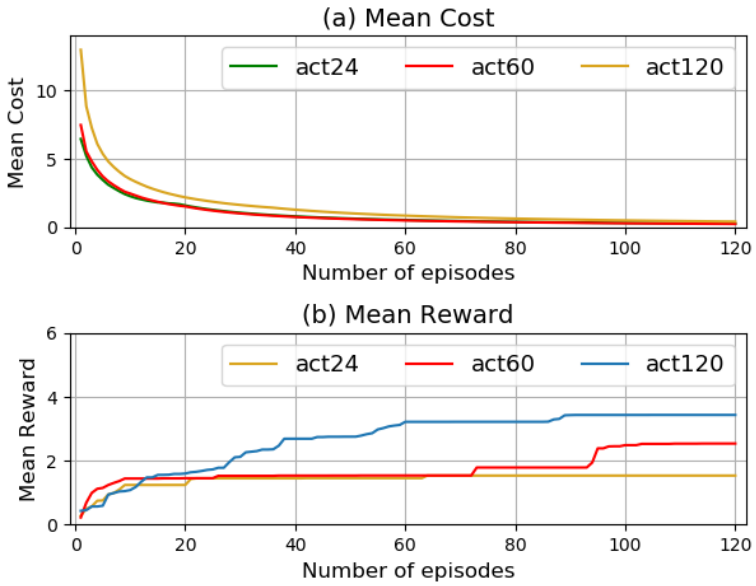

Fig. 4.13 Impact of varying the action space size while training the DQN agent

the more the reward function is increased. The increasing reflects the existence of appropriate new network configurations (i.e., network paths), so the queues of nodes could be unloaded and lead to decreasing both packet loss and network latency. As a result from this experiment, we decide to take the action space size as 120.

As mentioned in the equation (4.4), the reward function is related to Data rate (W), Network latency (L) and Packet Loss (PL) with parameters α, β and γ, respectively. These parameters play an important role to determine, in one side, the importance of each factor and on the other side the convergence of the Loss function. To this end, we plot in Fig. 4.14 the evolution of the average of the Loss function under different number of training episodes, while changing the reward function parameters. From Fig. 4.14 we can see that, the less we give importance to both packet loss and network latency, the more the Loss function converges and gives small values. As a result, we decide that the best DQN model corresponds to the one with action space size 120, and the reward function parameters ($\alpha = 0.3, \beta = 1, \gamma = 1$). In the next experiments, we assess the performance of our proposed heuristic while using the aforementioned DQN model and prediction methods.

In order to evaluate the performance of our proposed solution DTPRO, which corresponds to combining the best obtained DQN and LSTM models, we compare it with the following baselines:

- Hop-count (HC) based routing, which is the default routing metric used by ONOS.
- Reduced DTPRO by using the obtained DQN model for routing optimization and at the same time disabling the Traffic Prediction module.
- DTPROv1, which consists in using the obtained DQN model for routing optimization and the obtained ARIMA model (instead of LSTM) for Traffic Prediction.

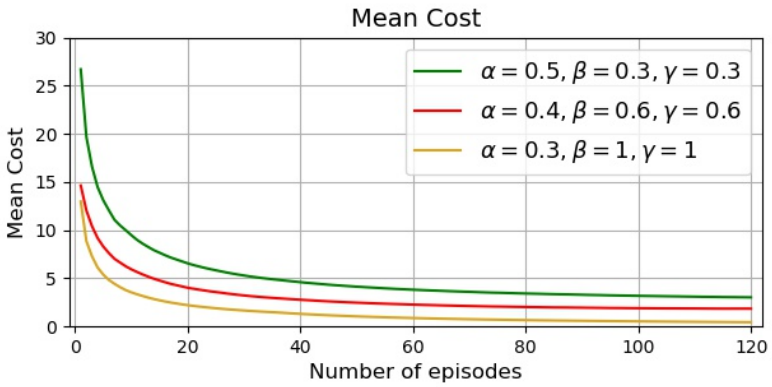

Fig. 4.14 Impact of varying the reward function parameters α, β, γ while training the DQN agent

- DTPROv2, which consists in using the obtained DQN model for routing optimization and LR for Traffic Prediction.

Fig. 4.15 plots the packet loss, the delay and the link utilization for all schemes (DTPRO, DTPROv1, DTPROv2, Reduced DTPRO, and HC). We can see that the HC approach causes obviously considerable packet loss and increases the link utilization, since all the flows are forwarded to shortest paths that shares the same link with minimum capacity. On the other hand, when using the DQN agent without prediction (i.e., the Reduced DTPRO scheme), considerable packet loss is still observed, increasing thus the link utilization, due to the incapacity of DQN to predict the future evolution of network traffic. Finally, we can see that DTPRO outperforms all other schemes, especially DTPROv1 and DTPROv2, where the packet loss, delay and link utilization are decreased. This is related to the high accuracy of LSTM in predicting network congestion, compared to ARIMA and LR prediction methods.

We plot in Fig. 4.16 the number of predicted congestion, while combining the DQN with different traffic predictions methods. It is worth noting that, the congestion happens when the network latency or traffic load exceed a certain threshold which is fixed to 80% of each link capacity. In the proposed network traffic evolution from 3000 states defined above in this section, we force certain states to overload the network for some specific network configurations. From this figure, we can see that LSTM outperforms others methods due to its high accuracy for predicting the future evolution compared to ARIMA and LR.

To improve the Quality of Experience (QoE), it is necessary to provide wider varieties of services than just a single class of best-effort service [141]. To this end, we propose to evaluate our DTPRO approach with and without *priority* according to the following simulation scenario: we use initially specific paths for Latency Sensitive Application (LSA), Throughput Sensitive Application (TSA) and Packet Loss Sensitive Application (PLSA). Then, we force these paths to be delayed and congested. Based on a specific set of priorities (p_1, p_2, p_3), Fig. 4.17 compares the following baselines:

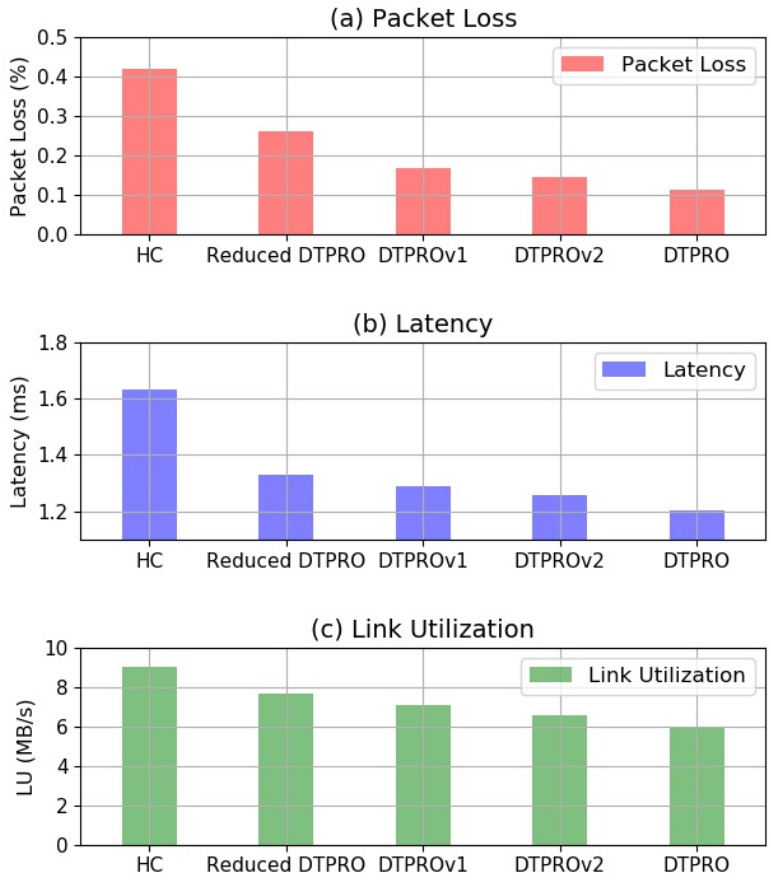

Fig. 4.15 Packet Loss, Delay and Link utilization under rule placement algorithm based on DQN with and without prediction

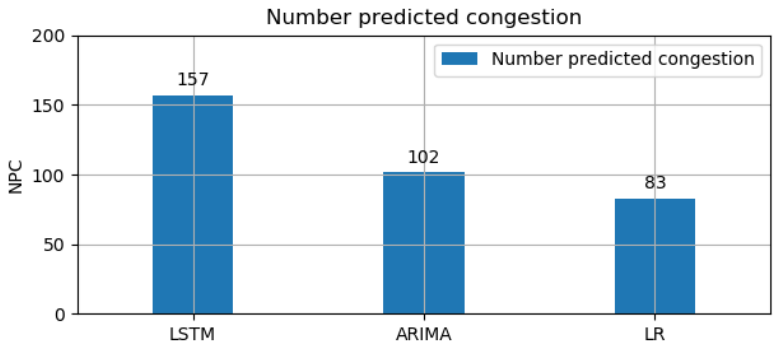

Fig. 4.16 Number of predicted congestion while combining the DQN with different traffic predictions methods

- DTPRO_Flow_Size corresponds to the proposed DTPRO heuristic which sorts the traffic flows according to their size to reroute the traffic once there is a congestion.

- DTPRO_Priority corresponds to the proposed DTPRO heuristic which sorts the traffic flows according to their *priority* to reroute the traffic once there is a congestion.

The set of priorities are selected as follows:

- p_1: $p(LSA) = 10$, $p(TSA) = 5$, $p(PLSA) = 1$
- p_2: $p(LSA) = 5$, $p(TSA) = 10$, $p(PLSA) = 1$
- p_3: $p(LSA) = 1$, $p(TSA) = 5$, $p(PLSA) = 10$

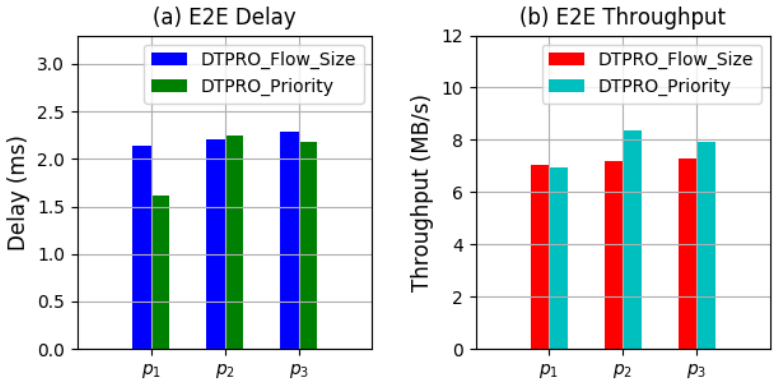

Fig. 4.17 DTPRO with and without priority

From Fig. 4.17 (a) we can see that, when giving more priority to LSA in p_1, DTPRO_Priority performs better than DTPRO_Flow_Size in decreasing the E2E Delay. However, when giving more priority to TSA in p_2 and PLSA in p_3 the E2E Delay for both schemes is close to each other. On the other hand, we can see from Fig. 4.17 (b) that, when giving more priority to TSA in p_2 or PLSA in p_3, DTPRO_Priority performs better than DTPRO_Flow_Size in decreasing the E2E Throughput. However, when giving more priority to LSA in p_1, the performances for both schemes are close with no improvement in the E2E Throughput. The reason of improving the performances is that the flows corresponding to these application types in the congested paths are routed first. This allows the DTPRO approach to be used in a context of Network Slicing where each slice is dedicated for specific traffic types.

Fig. 4.18 plots the impact of varying the T_{DQN} time interval on the network performance (i.e., Number of Predicted Network Congestion, the E2E Delay and the E2E Throughput or Rate). This parameter is varied in the set of intervals: [1h, 2h, 4h, 6h, 8h, 12h, 24h]. Recall that, the T_{DQN} parameter corresponds to the time interval to apply a new routing strategy obtained from the DQN agent.

From this figure we can observe that both the Network Congestion and E2E throughput increase with the increase of the time interval T_{DQN}. On the other hand, the E2E Delay decreases with the increase of T_{DQN}. The reason is that the DQN state space increases with the increasing of the T_{DQN}

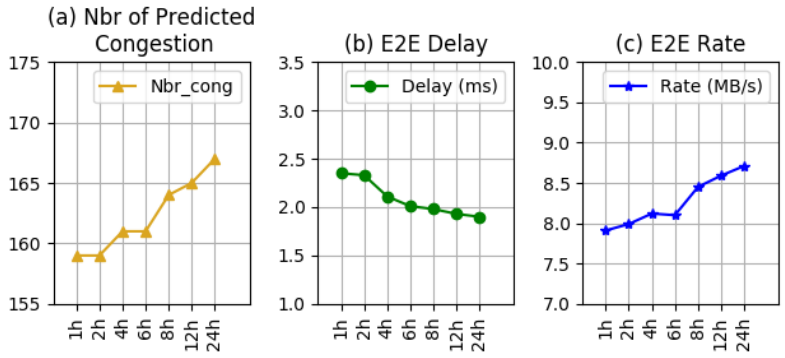

Fig. 4.18 Impact of varying the T_{DQN} time interval on the network performance

time interval, where new transitions and new network configurations are detected. Moreover, the LSTM model in the Traffic Prediction module is able to predict new traffic evolutions, which leads to predict new network congestion and rerouting the traffic more efficiently, decreasing thus the E2E Delay and increasing the E2E Throughput.

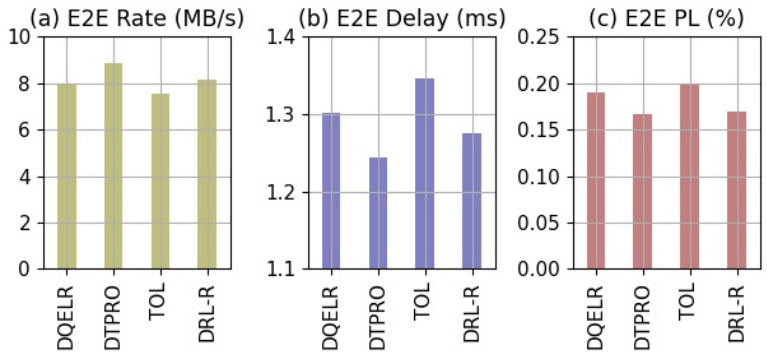

Fig. 4.19 Comparative analysis between DTPRO, TOL, DRL-R and DQELR

Finally, Fig. 4.19 shows a comparative analysis between the proposed approach DTPRO and three main approaches indicated in the related works: 1) Deep Reinforcement Learning-based Routing (DRL-R) [96], which represents a traffic optimization solution based on DQN, 2) Traffic Optimization based on LSTM (TOL) [22], which corresponds to the use of the LSTM RNN framework for predicting the network traffic matrix, and 3) Deep Q-Network-based Energy and Latency-aware Routing (DQELR) [143], which is a routing optimization solution based on DQN.

As the design of DTPRO is modular, in which the knowledge plane consists of two separated modules: i) routing optimization based on DQN and ii) Traffic Prediction based on LSTM, the comparison of our proposed approach with DRL-R and DQELR is done by deactivating the traffic prediction module and using only the routing optimization module. On the other hand, for the TOL approach, we used only the Traffic Prediction module.

We used the same experimental topology, shown in Fig. 4.7, for all approaches. For DRL-R [96], the DQN Q-Network and Q-Target consist of two hidden layers with 30 neurons. The Relu corresponds to the activation function and the learning rate is fixed to 0.001, as proposed in [96]. For the DQELR model [143], the input layer consists of four nodes. Three hidden layers with 300, 150 and 15 nodes, respectively, are also used. Finally, the TOL approach consists in using the LSTM model in [22]. All these models are trained based on the aforementioned parameters. Then, we measured the E2E Delay, the E2E Throughput and the E2E Packet Loss for each approach for a time interval of 24h, as shown in Fig. 4.19.

From this figure, we can see that the TOL approach causes obviously more packet loss and delay, and provides less network throughput compared to all other approaches. This can be explained by the fact that, this approach does not take into account the best routing strategy, increasing thus the number of congestion links and leading to a new network behavior with low prediction accuracy. Second, we can see that DRL-R and DQELR perform better than TOL since both approaches consist in defining routing strategies by training DQN agents, while maximizing certain rewards related specifically to network throughput and delay. On the other hand, DRL-R performs slightly better than DQELR, since adding DQN based routing decisions to packets in the DQELR approach impacts the performances and decentralizes the routing decisions, which is contradictory to the SDN design. Finally, the superiority of our proposed DTPRO method over all other approaches, in terms of E2E Throughput, E2E Delay and E2E Packet Loss, is clearly visible. The reasons are: i) different from DRL-R and DQELR, the action of our proposed DQN model is to modify the links weights vector, which is equivalent to defining all flow paths in one action instead of defining the path for each incoming flow, as defined in DRL-R and DQELR, ii) the TOL approach predicts only the traffic matrix without considering the routing strategy, and iii) combining the DQN agent with the traffic prediction based on LSTM allows unseen transitions from the DQN agent to be predicted by the traffic prediction module by considering the previous experiences in the DQN agent and at the same time the future behavior of network traffic in LSTM.

4.4 Conclusion

In this chapter, we presented a method for rules placement in Software-defined Networks based on real-time statistics measurement and traffic prediction, which are implemented separately as a cooperating modules on both the Control Plane and the KP layers. By taking advantage of the KP, the network routing is dynamically optimized by deploying a DQN agent that dynamically determines the optimal policy mapping the set of states (Traffic Matrices) to the set of actions (changing the vector of link weights). In addition, we proposed to deploy a Traffic Prediction module based on the well known prediction methods LSTM, in order to avoid congestion. To this end, we have mathematically formulated the QoS-aware routing problem as a LP, where the corresponding optimization problem is to minimize the total network latency, packet loss and link utilization. To solve this optimization problem, a simple yet efficient heuristic algorithm was proposed and implemented, called DTPRO

that dynamically interacts with the external DQN agent module to get the set of link weights, and the Traffic Prediction to avoid congestion. Experimental results using the ONOS controller and OVS, showed that the DQN agent is able to learn a mapping between the Traffic Matrix state and the set of link weights to route the traffic flows. However, DQN itself is not well adapted for predicting the future evolution of the network traffic. By combining DQN with Traffic Prediction, we showed that network latency, packet loss and link utilization can be decreased. Moreover, we showed that LSTM achieves a high estimation accuracy, which outperforms traditional prediction methods, and decreases both E2E delay and packet loss.

The next chapter will leverage the proposed architecture as well as the corresponding ML methods in an E2E network slicing of a 5G platform.

Chapter 5

Online based learning for predictive end-to-end network slicing in 5G networks

Contents

5.1 Introduction

NFV [59] and SDN [116] are the key complementary technologies for the 5G vision, allowing together high flexible and programmable radio and transport network. NFV decouples Network Function (NF) from hardware resources and provides flexibility by dynamically instantiating a VNF as a virtual machines running on top of servers, switches or routers. On the other hand, SDN decouples the control plane from the underlying infrastructure and assigns the control logic to a centralized controller. These new technologies allow the providers to control and orchestrate their resources and enable the sharing of the same physical network infrastructure through the concept of Network Slicing [19].

Network slicing enables optimal support for heterogeneous services, by running them in an independent Virtual Networks (VN)s, on a common shared Physical Network Infrastructure (PNI), such that each VN is allocated a fixed or dynamic portion of the PNI resources. In the most general case, network slicing is generally applicable in both RAN referred to as radio slicing and transport and core network domains, referred to as transport network slicing. Radio slices share radio resources among diverse services (i.e., IoT, eMBB) by reserving an appropriate portion of bandwidth (i.e. a number of RBs) to be used from that slice for a specific time interval [47]. Transport network slices consist in creating isolated PN composed of physical or virtual switches or routers assigned to a tenant by the network provider that, in turn, owns the corresponding physical resources [80].

Creating an isolated and efficient end-to end network slice remains challenging, particularly when considering the bandwidth prediction of radio slices to scale in and out the PN slices, as well as congestion inside each PN slice taking into account the slices QoS.

To this end, we design and implement, in this chapter, an SDN based architecture for E2E network slicing which proactively and dynamically adapts radio slices to the transport network slices. The developed architecture enables the creation, modification and continuity of radio and transport network slices while considering their resource and QoS requirements. It leverages machine learning for predicting radio slices capacities, improving network resource utilization, and predicting congestion within each network slice. We also formulate this network slicing problem as a LP aiming to minimize the total network delay. Finally, we propose an efficient heuristic algorithm with low time complexity and high estimation accuracy to solve large problem instances. In addition, we validate our proposal using an experimental C-RAN prototype, which makes use of OAI [113], an open-source software implementation of LTE, spanning the full protocol stack of E-UTRAN and EPC [2], the ONOS SDN controller [26], and the SD-RAN FlexRAN controller [63].

The main contributions of this chapter can be summarized as follows:

- First, we design and implement a Northbound SDN application on top of ONOS and FlexRAN controllers to enable dynamic creation of E2E network slices (i.e., eMBB, mMTC), while considering the corresponding QoS.

- Second, we propose a ML based algorithm to predict the capacities of radio slices and adapt them to the PN slices in order to improve network resource utilization.
- Third, we consider the network slicing problem as a flow rules placement problem and formulate it as a LP aiming to minimize the total network delay. We then propose an efficient heuristic algorithm with low time complexity and high estimation accuracy to solve large problem instances.
- Forth, we validate our proposal using an experimental prototype.

The remainder of this chapter is organized as follows. In Section 5.2, we discuss the architecture of our framework and the proposed rules placement algorithm. Section 5.3 presents the environmental setup and evaluates the proposed method. We finally conclude this chapter in Section 5.4.

5.2 SDN-based dynamic network slicing continuity and traffic prediction

In this section, we detail our approach for SDN-based dynamic network slicing continuity and traffic prediction to enable efficient E2E slicing. Firstly, we explain the overall system architecture. Thereafter, radio slicing, PN slicing, E2E slicing and proactive E2E slicing will be described.

5.2.1 SDN-based dynamic network slicing continuity and traffic prediction Framework

Figure 5.1 presents the system architecture considered in this chapter, which consists of two interconnected technology domains. The first is the radio domain, which provides mobile broadband and IoT services. IoT Gateways connect IoT sensors/devices to the IoT platform in the cloud through the C-RAN network, so that data can be sent back-and-forth periodically between them. According to the C-RAN concept [1], i.e., the RAN functionalities are split between RRU, which includes a remote radio transceiver, that is interconnected via a Front-Haul (FH) interface to the RCC node. Moreover, we deploy C-RAN following an SD-RAN architecture, in which the RAN control and data planes are separated using a Radio Controller. This architecture offers significant benefits to the RAN, including the global view of the network state allowing us to dynamically handle slicing process. As will be detailed in the subsection 5.2.2, the radio slicing is performed by a northbound SDN application which communicates with the Radio Controller through the northbound API (Northbound Interface (NBI)) as shown in figure 5.1.

The second is the transport domain, which provides connectivity services to the Radio domain. This domain consists of SDN devices (switches, routers), that we modified to handle the GTP traffic, referred to us PN that connects the RAN to the core network. More specifically it connects the RCC to: a) the MME through the S1-MME interface and b) the SGW through the S1-U interface,

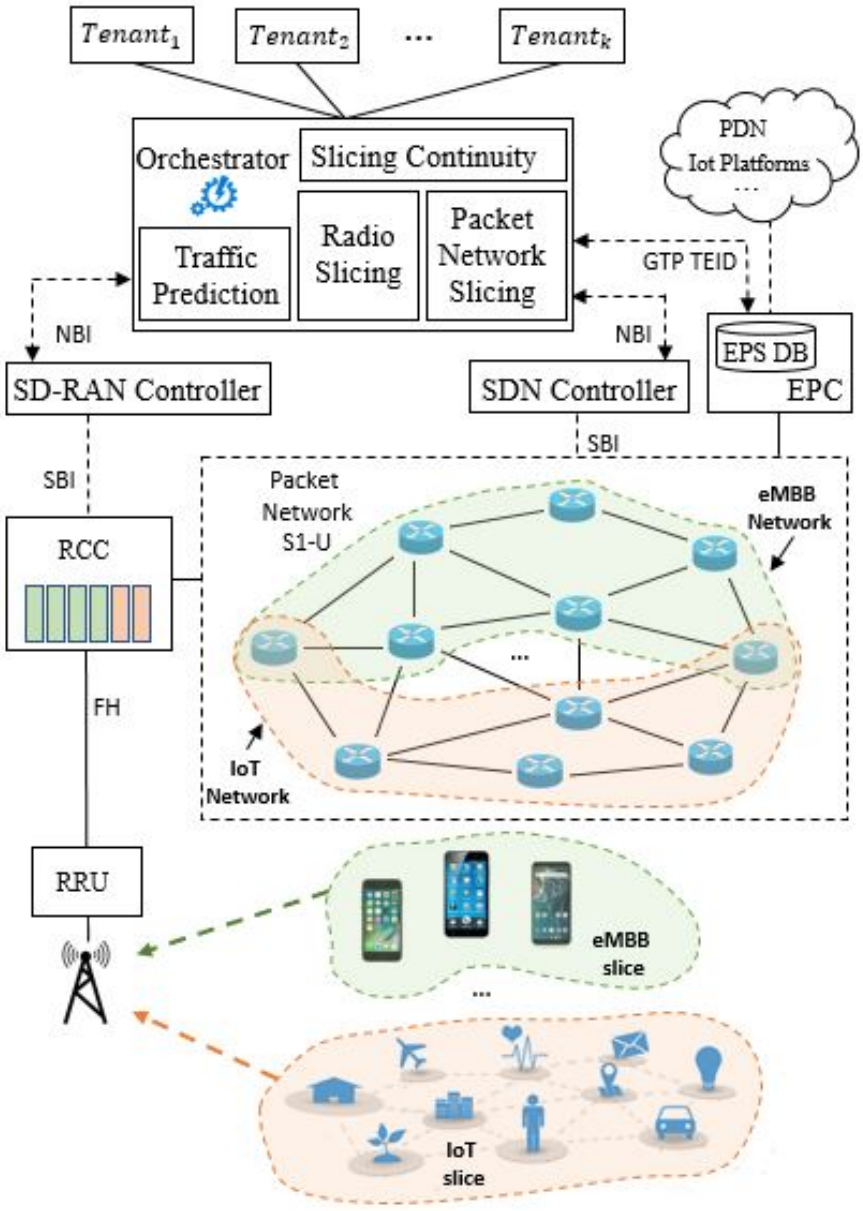

Fig. 5.1 SDN-based dynamic network slicing continuity and traffic prediction Framework

where the traffic can be routed through multiple paths. Moreover, in order to efficiently handle PN slicing, we centralize the control plane using an SDN Controller on top of these devices. The interface between the SDN Controller and the devices is defined by the OpenFlow protocol [101], allowing the instruction of devices on how to handle incoming UEs data packets. It is worth noting that, the PN slicing, as will be detailed in the subsection 5.2.3, is performed by a northbound SDN application, which requests the SDN Controller through the northbound API (NBI) to install flow rules corresponding to UEs packets.

In the considered architecture, the radio and PN slicing are performed by the network provider with the help of an orchestration layer that periodically collects statistics about data planes on both radio and PN slices from the SDN and SD-RAN Controllers, respectively, and performs slicing creation and modification. Furthermore, these statistics can be exposed, through the Orchestrator's NBI, to tenant applications. It is worth noting that, the orchestration of resource slices provisioning between tenants is outside the scope of this work. We are rather focusing on the slicing continuity, predicting radio slices capacities and traffic load balancing in the PN slices. In what follows, we detail these properties.

5.2.2 Radio Slicing

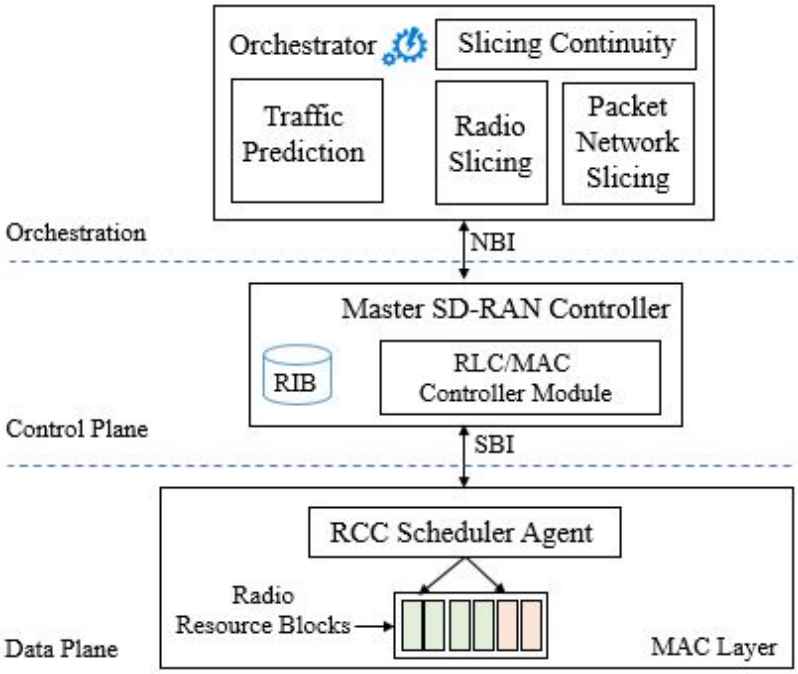

Fig. 5.2 Radio Slicing Architecture

Figure 5.2 shows the proposed radio slicing architecture. We distinguish between three layers. The first one is the orchestration layer, which consists of a radio slicing module. This module communicates with the SD-RAN Controller through the northbound API and dynamically performs slicing based on statistics collected and stored in the RAN Information Base (RIB). In this architecture, the slice creation requests come from the tenants for subscribers located in a specific area. Then, the Orchestrator decides whether the requested amount of resources can be allocated, while guarantying the slices QoS and SLA. Through the northbound API, the Orchestrator can create radio slices by setting a set of parameters, such as i) *slice_percentage* which corresponds to the number of RBs that the corresponding slice is allowed to use, as a fraction of the whole bandwidth, ii) *slice_duration*

representing the number of transmission time interval (TTI), and iii) *slice_QoS* which corresponds to the QoS requirements to satisfy subscribers demands such as the required throughput.

The second layer is the Control plane, which is represented by an SD-RAN Controller, performing the slicing scheduling algorithm. This algorithm can be decomposed into two parts: the control part that makes the decisions for the radio link, which is located in the SD-RAN Controller and the action part that is responsible for applying those decisions, which is located in the data plane. For example, the modulation, the coding scheme and the RB allocation, are scheduling decisions that have to be performed by the control part.

Finally, the Data plane layer, composed of the RCC, which is freed from the control responsibilities and handles only the decisions made by the SD-RAN controller, by means of a RCC scheduler agent. This latter defines a set of functions, through a specific southbound interface (SBI), to apply MAC scheduling decisions for RB allocation as well as request and obtain statistics such as transmission queue sizes of UEs, etc.

5.2.3 Packet Network Slicing

In the current EPS architecture, the traffic flowing from a RCC can be routed only through one single path in the S1-U interface, even if there are multiple paths. Since data are transported with GTP protocol, where the same outer headers (Ethernet, IP and Transport headers) are added to UE packets, SDN based routing of these packets using the information encapsulated on these layers (IP, MAC addresses, etc) through different paths is not possible. On the other hand, the UE connections can be identified by a specific field TEID of the GTP header encapsulated on top of these layers. The idea is thus to deploy a set of bridges capable of routing GTP traffic based on TEID identifiers, in order to provide more flexibility and visibility over the mobile network data traffic.

To this end, we have extended the open source implementation of OVS, to support the GTP forwarding feature. In fact, OVS is a multi-layer software switch which uses OpenFlow protocol to connect with Controllers. It consists of two main components, the first one is implemented on the kernel $openvswitch.ko$ and the second one is implemented on the user-space part $ovs\text{-}vswitchd$. In such a way, when a GTP packet is captured by an OVS Datapath it will be processed by the kernel module and skip the legacy Linux network stack. By taking this into account, we have extended the kernel module to be able to extract informations from the GTP header, in particular the TEID field, and the high layers (i.e., IP, UDP port, etc). Based on preinstalled GTP rules, we will be able to forward directly the GTP packets to a specific Datapath, without sending the packet to high layers. Doing so, the GTP flow rules are installed remotely following the SDN paradigm.

A PN slice is hence composed of OVS switches and links capable of routing the traffic flow from a pre-created radio slice and ensure continuity, as will be explained in the next section.

5.2.4 E2E network slicing

Figure 5.3 illustrates our proposed procedures of E2E network slicing according to the SDN paradigm. It is composed of two main steps. The first one is Slices Creation, in which the tenant begins by querying the Orchestrator to create a slice while guaranteeing a specific QoS target. The Orchestrator triggers the creation of Radio and PN Slices by contacting respectively the SD-RAN and SDN Controllers. Thereafter, the Orchestrator associates the Radio and PN slice informations such as *slice_id*, the set of OVS devices and their corresponding links and ports, and save them in the centralized Database. After that, the Tenant triggers a request to the Orchestrator to add some subscribers, identified by their International Mobile Subscriber Identity (IMSI), to the created slice. The latter associates the slice informations to the user identifiers and save them in the Database. The second step happens when a new connection is established (i.e., after UE Initial Attachment and Bearer creation and exchange of UpLink and DownLink TEIDs between RCC et SGW). In this case, the SD-RAN triggers the Orchestrator of the new connection event. The latter then, triggers the SD-RAN to assign the connection IMSI to the corresponding Radio slice. Thereafter, the Orchestrator requests the UpLink and DownLink TEIDs, the RCC and SGW IPs and some informations about devices such as *port_in*, *port_out*, from the Database by communicating the new connection IMSI. The latter finally assigns the connection to the PN Slice while the traffic flows from a Radio Slice is continued to the PN Slice.

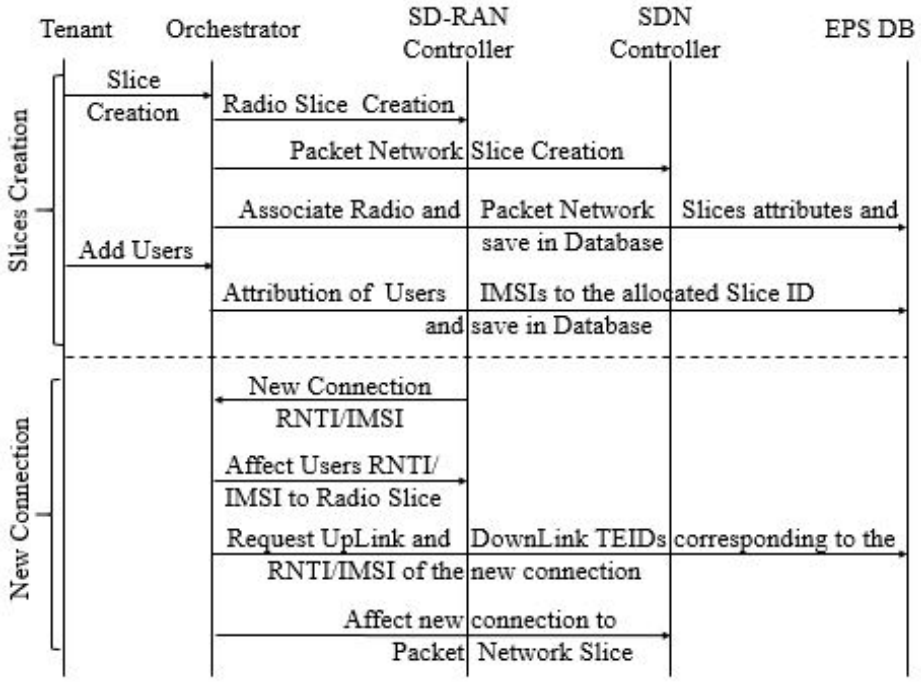

Fig. 5.3 E2E network slicing procedures

5.2.5 Proactive E2E Slicing

To meet the requested flow's QoS requirements, the sizes of Radio slices must be adjusted proactively and dynamically. To this end, we suppose that, initially, the number of resource blocks that the tenant's radio slice is allowed to use is fixed as a fraction of the whole bandwidth. However, we propose to proactively allocate additional unused resource blocks from other radio slices to the overloaded slices, which exceed their authorized capacity, by predicting the future evolution of resource blocks requirement, using the LR method [82]. On the other hand, these radio slices must be adapted proactively and dynamically to the PN slices. Furthermore, while scaling in/out the PN slices, we distribute the traffic load over the same slice topology. Then, to avoid congestion, we proactively route flows following the paths with less delay and large capacity. To this end, we build on our previous contribution proposed in chapter 4, in which we deployed two main modules on the SDN Controller. The first module is the Network Measurement that periodically collects statistics (i.e., latency, throughput, flow size, link/device state, etc.) from the data plane (i.e., PN slice) and stores them in a centralized Traffic Matrix. The second module is the Proactive Forwarding in which we aim to balance the traffic load over the slice topology. Based on the Traffic Prediction module deployed on the Orchestration layer, it avoids congestion by rerouting flows to the less delayed paths. This problem can be defined as a rules placement problem, where the objective is to determine which link to route which flow in order to minimize the total network delay and balance the network load by minimizing link utilization while scaling in/out the PN slice. We mathematically formulate this problem as a LP as follows:

- Inputs:
  - Network topology: $G(V_i, E_i)$
  - Traffic matrix: TM_i
  - Predicted Traffic Matrix: $\widehat{TM_i}$
  - Matrix of link delay: $D_i = \{d_{(u,v)}\}$
  - Matrix of predicted link delay: $\widehat{D_i} = \{\widehat{d}_{(u,v)}\}$
  - Matrix of link delay thresholds: $TS_i = \{TS_{(u,v)}\}$
  - Size of flow f allocated to (u,v): $S_{(f,u,v)}$
  - Link Capacity: $\forall (u,v) \in E$: $C_{(u,v)}$
  - Set of flows: $F = \{f_1, f_2, f_3, ..., f_l\}$
- Objective:
  - Minimize $\alpha.D + \beta.LU + \gamma.PL$
- Constraints:

- Delay limitation: $\forall(u,v) \in E_i$:
  $Max(d_{(u,v)}, \widehat{d}_{(u,v)}) \quad < \quad TS_{(u,v)}$
- Link capacity limitation:
  $\forall(u,v) \in E_i : LU_{(u,v)} < MLU_{(u,v)}$
- Path capacity limitation:
  $\forall(u,v) \in P$: $MLU_{(u,v)} - LU_{(u,v)} \leq Cap\_Av\_Path(\ P)$
- Demand satisfaction:
  $\forall(u,v) \in E_i, \forall f \in F : \sum S_{(f,u,v)} \leq RSD_i$

$$Scale\_Out_i = \begin{cases} 1 \ \ if\ \widehat{RSD_i} \ > \ TSD_i \\ 0 \ \ Otherwise \end{cases}$$

Where $MLU_{(u,v)}$ (Maximum Link Utilization of link (*u*,*v*)) denotes the adjusted maximum link capacity in order to avoid congestion. It is defined as follows:

$$MLU_{(u,v)} = c_{(u,v)} * \theta_{(u,v)} \tag{5.1}$$

where $c_{(u,v)}$ denotes the link capacity, and $0 < \theta_{(u,v)} < 1$ is a constant depending on link characteristics. Also, $LU_{(u,v)}$ determines the utilization of the link (u,v), which is defined as follows:

$$\forall f \in F : LU_{(u,v)} = \sum (E_{(f,u,v)} \times \ S_{(f,u,v)}) \tag{5.2}$$

where the symmetric Binary matrix $E = \{e_{(f,u,v)}\}$ denotes whether the flow rule f is allocated to the link (*u*,*v*) or not. The capacity and delay limitation constraints force each link (u,v) to not exceed its threshold MLU and to not be delayed. The demand satisfaction constraint ensures the PN slice to send or receive the same amount of traffic from the Radio slice and proactively scale in/out the PN slice if the predicted traffic exceeds certain threshold, where TSD_i and $\widehat{RSD_i}$ denote, respectively, the threshold and predicted demand of Radio Slice i. It is worth noting that the threshold TSD_i can be fixed or determined dynamically while predicting radio traffic.

The objective is to minimize the total delay D, the total link utilization LU and the total packet loss PL, where α, β, γ are parameters related to the degree of importance of D, LU and PL, the latter are defined as follows:

$$D = \sum_{(u,v) \in E} d_{(u,v)} \tag{5.3}$$

$$LU = \sum_{(u,v) \in E} LU_{(u,v)} \tag{5.4}$$

$$PL = \sum_{f \in F} (dem_f - \sum_{(u,v) \in E} S_{(f,u,v)}) \tag{5.5}$$

where the variable dem_f represents traffic demand incoming to the network and $\sum_{(u,v)\in E} S_{(f,u,v)}$ represents the traffic after traversing the network.

Solving the mathematically formulated problem in the Orchestrator for each incoming flow could be time consuming, as the size of the network and the number of flows increase. Consequently, the computational complexity increases exponentially. Therefore, such approach is not feasible in practice, since it generates high overhead due to the frequent updates of the flow tables.

To cope with this problem, and reduce the computation time and complexity, we propose a simple yet efficient heuristic algorithm, called Congestion prediction and Load balancing based Rule placement algorithm (CLR). CLR can efficiently find a feasible and near optimal solution, while minimizing the total network latency and packet loss.

Algorithm 2: CLR Rule placement algorithm

```
1:  procedure CONGEST_REM(G(V_i,E_i),TM_i,TM̂_i)
2:      for all link (u,v) ∈ E do
3:          t ← 0
4:          while Max(d_(u,v), d̂_(u,v)) > TS_(u,v) or LU_(u,v) > MLU_(u,v) do
5:              CP ← Get_Congested_Path(u,v)
6:              F ← Sorted_Flows(CP)
7:              R ← F[t].flow
8:              SP ← Sorted_Backup_Paths(R)
9:              for each path p ∈ SP do
10:                 if p can route R then
11:                     Install_Rule(p,R)
12:                     Remove_Rule(CP,R)
13:                     Break
14:             t ← t+1
15: end procedure
16: procedure LOAD_BALANCING(G(V_i,E_i),TM_i)
17:     for all flow f ∈ sorted F do
18:         list_P ← Paths that can route f in Slice_i
19:         for all P ∈ list_P do
20:             if Scale_Out_i then
21:               if mp(P) + f.size < mt then
22:                   Assign f to P
23:                   mt ← Max(LU_(u,v), ∀(u,v) ∈ E)
24:             if Scale_In_i then
25:               if mp(P) + f.size < MLU(P) then
26:                   Assign f to P
27: end procedure
```

Algorithm 1 shows the detail of the proposed heuristic. It consists of two procedures: CONGEST_REM and LOAD_BALANCING that take as inputs the Network topology $G(V_i,E_i)$ the current and predicted Traffic Matrix (TM and $\widehat{TM}$, respectively).

The CONGEST_REM procedure is called to delete a congestion and it works as follows. Upon measuring continuously the current and predicted Traffic Matrix, if a congestion occurs in the PN $Slice_i$, in which the current or predicted link delay is greater than a certain threshold or the link is overloaded, this algorithm finds the flow rule R corresponding to the flow with minimum size in the congested path CP. Then, it sorts all other paths (denoted by SP) by the delay matching the flow rule R (lines 5-8). In this case, the flow rule R must be rerouted to a path in SP (line 9-13). If no path in SP can accommodate the corresponding flow size, the flow is discarded, and the algorithm goes to the next flow in the CP (line 14).

On the other hand, the LOAD_BALANCING procedure triggered periodically and when scaling in/out a PN $Slice_i$, and it works as follows. For each flow f in F sorted in descending order, it evaluates possible paths $list\_P$ that can route the flow f. Among these considered paths, we distinguish two situations. The first one, when scaling out the PN $Slice_i$, in which we distribute the flows over the new paths, to this end we minimize the overall LU determined as follows:

$$mt = Max(LU_{(u,v)},\ \forall (u,v) \in E) \tag{5.6}$$

The second one happens when scaling in the PN $Slice_i$, in this case the path sizes increase and we distribute the flows over the original paths while avoiding overloading each link. Note that, $mp(P)$ denotes the maximum path utilization, it is determined as follows:

$$mp(P) = Max(LU_{(u,v)},\ \forall (u,v) \in P) \tag{5.7}$$

Whereas $MLU(P)$ denotes the maximum threshold determining the maximum tolerable load of all link composing P. It is determined as follows:

$$MLU(P) = Max(MLU_{(u,v)},\ \forall (u,v) \in P) \tag{5.8}$$

5.3 Performance Evaluation

In this section, we evaluate the efficiency of our proposed approach. We start by presenting the IoT platform, thereafter, the E2E setup. Then, we present the experimental results.

5.3.1 IoT Platform

Figure 5.4 shows the IoT Platform architecture, which consists of four components namely: connected things component, communication and identification component, computation component and services and applications component [17][72].

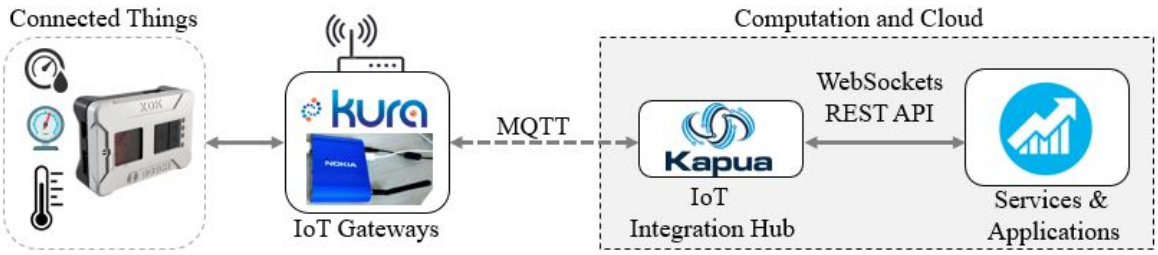

Fig. 5.4 IoT Platform (based on [17][72])

Connected things component

We make use of the Bosch XDK Sensor (XDK110) as the main element in this part of the IoT platform. The Bosch XDK is an all-in-one universal programmable sensor device and prototyping platform [30], which allows to measure a broad range of parameters such as humidity, pressure, temperature based on its embedded Micro-electromechanical systems (MEMS) sensors, which makes it suitable for divers IoT applications. The data collected by the sensors is converted and sent to specific destination in the cloud through a specific gateway.

Communication and identification component

The communication and identification component is a set of equipments and protocols that transport the collected data by the connected things to a remote computation platform as well as transporting configuration and instruction from the management system. The IoT gateway in particular connects a specific sensor when it is not possible to connect it via a specific protocol. Several protocols can be used to establish communication between the sensor and the remote management system. In this work, we make use of the Message Queue Telemetry Transport (MQTT) protocol, Nokia IoT Gateway and Eclipse Kura.

- **MQTT protocol:** is a messaging protocol implemented over TCP/IP to connect embedded devices in the connected things to networks with applications and middleware [50], it uses message broker server in the middle in communication between devices. Moreover, it supports Secure Sockets Layer (SSL) and TLS to transport sensing data.

- **Eclipse Kura:** is a Java/OSGi-based framework for building and running IoT gateway Machine-to-Machine (M2M) applications [124]. It provides a specific driver that can be used to interact with Bosch XDK110 devices. Based on this, we make use of the Nokia IoT Gateway that supports the new Bluetooth LE APIs and has this driver. In this way the XDK Workbench tool [156] can be used to implement applications that collect data from the XDK110 devices. Then, this application can be embedded to the Nokia IoT Gateway which in turn forward the collected data to the corresponding database or management system such as Eclipse Kapua, by the mean of the MQTT protocol.

Computation component

The computation component represents the processing unit which provides the computational capability of the IoT platforms, which is located in the cloud to process the data collected from sensors and devices. In this way, we make use of Eclipse Kapua [55] which is an open-source modular framework for IoT-Cloud integration, which is based on Java OSGi and message brokering. It provides connection to the IoT devices through IoT gateways by the mean of the MQTT protocol.

Services and applications component

The Services and applications component represents the brain of the IoT platform, which provides services, based on data collected by the sensors and devices, such as data collection, analytics, monitoring, management and security. It is interconnected with the Computation component via MQTT or WebSockets.

5.3.2 E2E Architecture

Figure 5.5 illustrates the logical building blocks of our framework, which consists of two interconnected technology domains. The first one is the radio domain deployed according to an experimental C-RAN infrastructure [1], in which the baseband processing functions are carried out at the RCC node that is interconnected via a FH to the RRU node. By making use of OAI software [113] and Docker container technology [52], we deployed the RRU and RCC on separated physical machines. The physical machine hosting the RRU container is connected to an Ettus Universal Software Radio Peripheral (USRP) B210 card [151] via an USB 3.0 interface. Note that we used 20 MHz channel bandwidth for our USRP B210 cards, which means that the total available RBs is equal to 25. The C-RAN infrastructure provides 4G connectivity to two Nokia IoT Gateways, one 4G Android smartphone and Bandluxe C501 LTE modem. The Nokia IoT Gateways in turn provide Bluetooth Low Energy (BLE) connectivity to Bosch XDK110 sensors. Note that, by making use of XDK-Workbench [156] tool, we developed a software for XDK senor that periodically sends Humidity, Pressure, Temperature, through the IoT Gateway by the mean of MQTT protocol [50], to a remote IoT framework. The latter is deployed based on Eclipse Kapua [55] on a remote machine as a cluster of Docker containers, as demonstrated in figure 5.4. Also, we connect a computer to the Bandluxe C501 LTE modem in order to generate UpLink and DownLink traffic. The second domain is PN, which consists of a set of OVS devices, created by deploying, on a physical machine, a specific OVS version that we extended to support GTP traffic. This PN is connected to the core network EPC, where the EPC functions are deployed in a cluster composed of two physical machines by the means of Docker container technology. The EPC contains the PGW function serving as a Gateway to the external networks such as Internet and Kapua IoT Platform.

In order to enable remote control of the OAI MAC Layer, we deployed the SD-RAN FlexRAN controller [63] on top of RCC function. We deployed the OpenFlow controller ONOS [26] to control

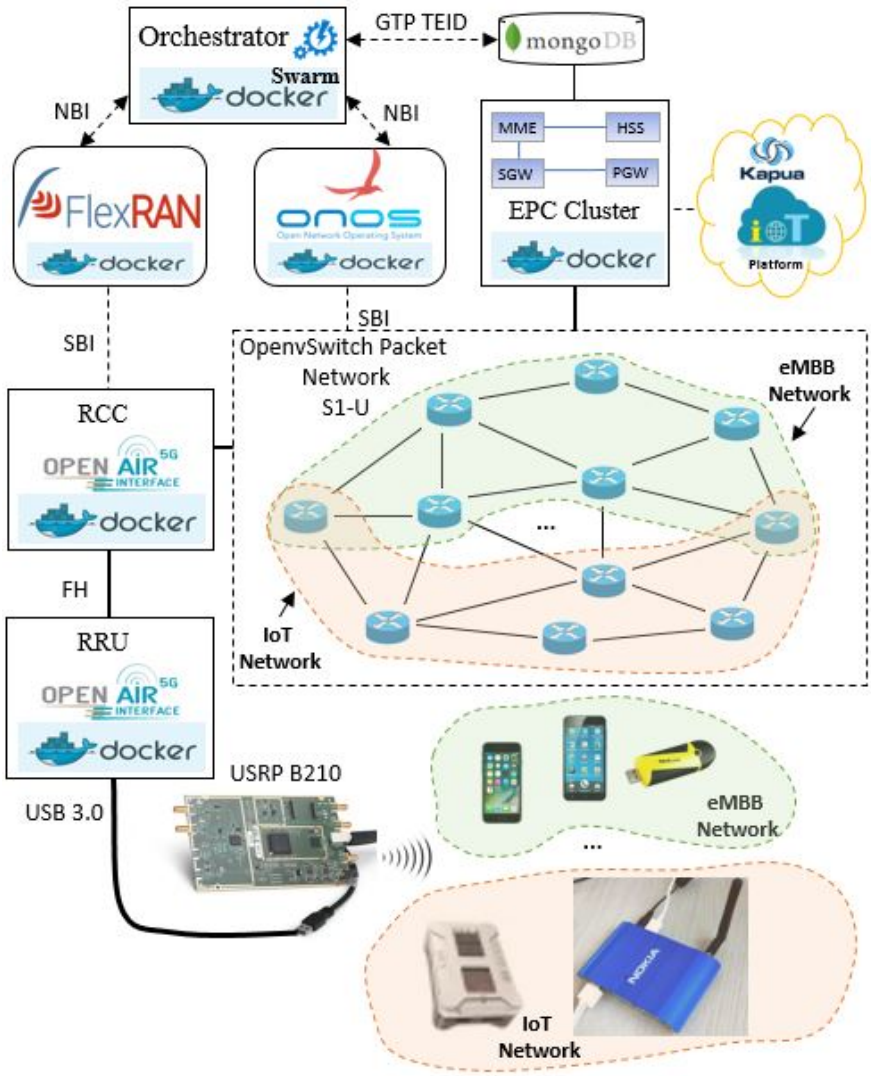

Fig. 5.5 E2E Slicing Prototype

the PN. On top of these two Controllers the Orchestrator is deployed by using Docker. The Radio Slicing module, the PN slicing module, the Slicing continuity module, and the Traffic Prediction module are developed based on Python [7], which interact with SD-RAN and SDN controllers through the FlexRAN and ONOS northbound API, respectively. Furthermore, the statistics needed to enable automation of slicing continuity such as GTP TEID, are extracted from the MongoDB database [5]. It is worth noting that, we implement the Network Measurement module (Latency Measurement and Statistics) as well as the Proactive Forwarding module (the module deploying the proposed heuristic) as cooperating modules for the Java based OpenFlow controller ONOS, based on our previous framework developed in chapter 4.

It is worth noting that, the main prediction method used in the proposed solution is LSTM. Therefore, we built and trained the LSTM model by using Keras Library [4], where the number of

dense layers is 2 and the number of nodes is 42. Note that, in order to improve the performances of our system, we saved the trained LSTM model, in such a way that only one step is needed to get the predicted traffic from the LSTM. In parallel, these models continue to learn from these new steps. Furthermore, it is expected that several RCCs share the same physical transport network through different PN slices. To this end, and to show the capacity of our proposed heuristic in predicting congestion and balancing the traffic load among the set of available paths, we have added an external emulated IoT and eMBB traffics to the PN part.

As for ONOS and FlexRAN controllers, the radio nodes (i.e., RRU, RCC) and the core network nodes (i.e., SGW, PGW, MME, HSS) are Docker based deployment. They are placed and managed by the containers clustering and scheduling tool Docker Swarm [53].

5.3.3 Experimental results

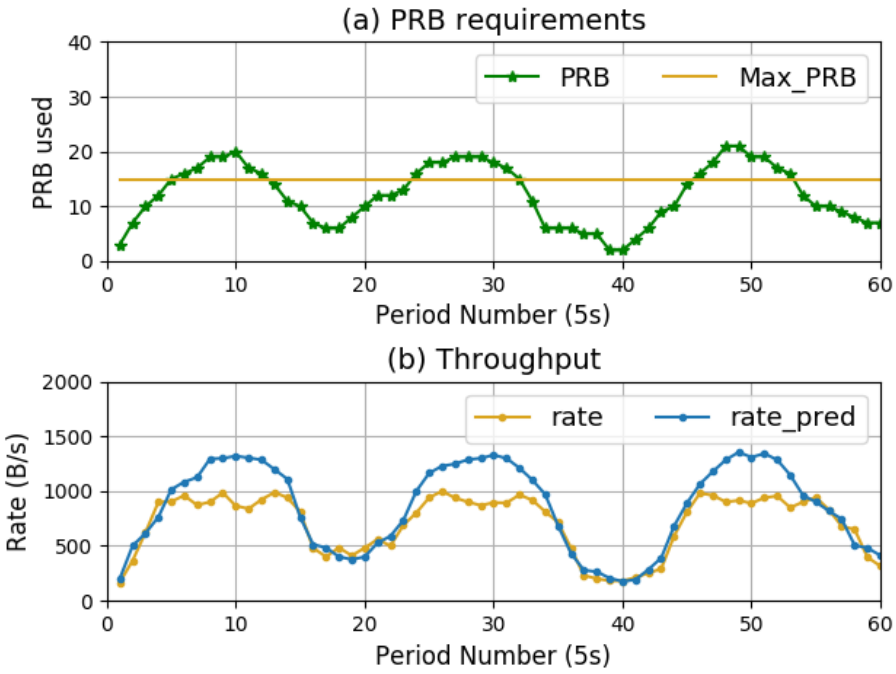

Fig. 5.6 IoT traffic profile and fixed and dynamic proactive radio slicing

In order to evaluate the performance of the proactive and dynamic radio slicing, we compare it to the radio slicing without prediction. Specifically, as shown in figure 5.5, we created two radio slices (i.e., IoT and eMBB). Then, we generated two Iperf traffics from two hosts. The first one is connected to the Bandluxe C501 LTE modem and the second one is connected to the IoT Gateway. Initially, we allocate 60% of the total RBs to the IoT slice and 40% of the total RBs to the eMBB slice. Thereafter, we generated an Iperf traffic from the IoT slice characterized by periodical sessions

which can exceed the slice capacity. On the other hand, the traffic generated in the eMBB slice is regular which takes regularly 70% of the RBs allocated to the eMBB slice. Figure 5.6 (a) shows the amount of RB requested by the IoT slice when all the bandwidth is allocated to this slice, and the maximum RB that this slice is allowed to use (i.e., 15 RB), during a period of 60 time intervals of 5 seconds. From figure 5.6 (b), it can be observed that the throughput is increased while predicting the future behavior of the traffic, and allocating additional RB not used from the other slice. However, the throughput remains stable at certain threshold, by using fixed slices capacities, since there is no dynamicity and the absence of future vision on the traffic evolution, which causes service degradation and packet loss.

To show the accuracy of our prediction method LR used to predict the RB allocation, we plot in figure 5.7 the MSE [22], calculated as follows:

$$\text{MSE(LR)} = \frac{1}{D}\sum_{i=1}^{D}(RB_i - \widehat{RB}_i)^2 \tag{5.9}$$

where RB_i *and* $\widehat{RB}_i$ are, respectively, the real and predicted RBs computed at time interval i (which is fixed to 5 seconds) and D corresponds to the total number of time intervals of 5 seconds (which is fixed to 60 time interval). From that figure, we can see that LR achieves a good estimation accuracy since the associated MSE values remain very low.

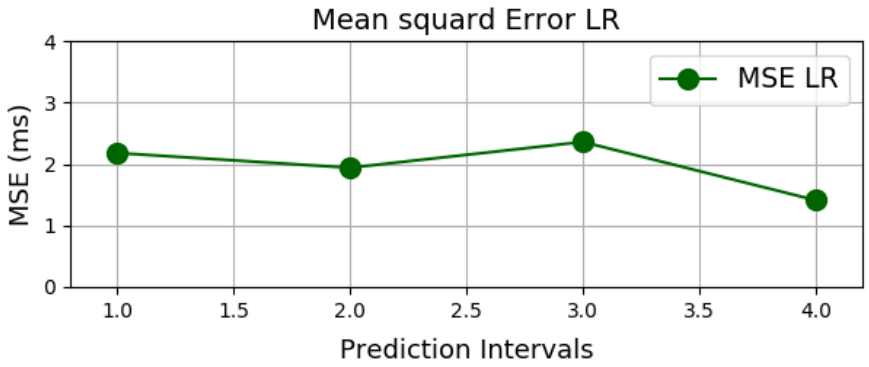

Fig. 5.7 MSE of LR used to predict RB requirements

In order to evaluate the proposed E2E network slicing solution, we propose to run it under two main strategies: i) Static, in which the amount of resource blocks and the capacity of PN slices are fixed and no scaling in/out, and ii) Zero_touch, where both the Radio and PN slices capacities are changed dynamically and proactively by the Orchestrator. In this latter strategy, we distinguish between the following schemes:

- CLR, which corresponds to the proposed heuristic while using LSTM for Traffic Prediction in PN slices and LR in the corresponding Radio Slices

- CLRv1, which corresponds to CLR while using LSTM for Traffic Prediction in PN slices and no prediction in the corresponding Radio slices.
- CLRv2, which corresponds to CLR while using LR for Traffic Prediction in PN slices and LR in the corresponding Radio Slices.
- HC based routing, which is the default routing metric used by ONOS, while using LR for Radio traffic prediction.

Note that the action of scaling in/out after predicting the radio traffic consists in adding or removing a set of paths in the PN slice in order to improve the performances.

First, we compare in figure 5.8 the prediction accuracy of the different methods used for traffic prediction (i.e., LSTM and LR) inside a PN slice for the eMBB traffic, by using the current and predicted link delays in the MSE. From that figure, we can see that LSTM achieves a good prediction accuracy, due to its capacity to learn long-term dependencies. On the other hand, we can observe that LSTM performs better than LR. The reason is that LR is not well adapted to random behavior.

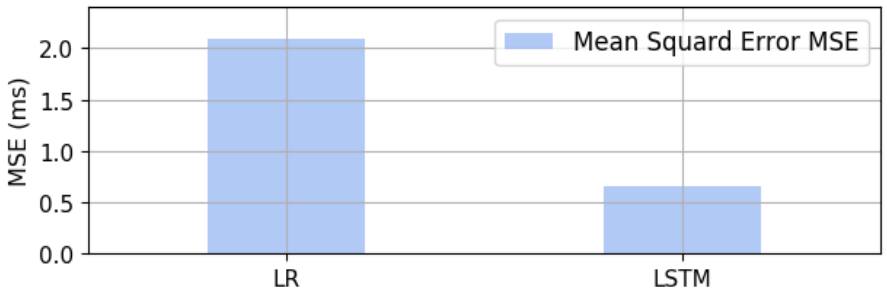

Fig. 5.8 MSE of CLR using LR and LSTM prediction methods

Figure 5.9 plots the E2E delay, packet loss and throughput for all schemes (CLR, CLRv1, CLRv2, HC and Static). We can see that the Static approach causes obviously considerable packet loss and decreases the throughput, since the slices capacities either in Radio or PN are fixed in advance. In addition, there is no scaling in/out. On the other hand, the Zero_touch approaches outperform the Static one, since the capacities associated to each slice are dynamic, based on traffic prediction and PN scaling. Specifically, HC outperforms the static approach since the Radio slicing is dynamic. However, it still causes considerable packet loss and decreases the throughput and latency, since all the flows are forwarded to shortest paths which are not always the optimal ones. On the other hand, the two approaches CLRv1 (i.e., using LSTM to predict congestion and no Radio prediction) and CLRv2 (i.e., using LR to predict congestion in both PN and Radio domains), improve the network performances in terms of decreasing the packet loss and increasing the throughput. However, a certain level of packet loss is still observed in these two schemes due to the incapacity of LR to predict the future evolution of network traffic in CLRv2 and the absence of traffic prediction in CLRv1. Finally, we can see that CLR outperforms all other schemes, where both the delay and packet loss are decreased,

and the throughput is increased. This is related to the high accuracy of LSTM in predicting network congestion, compared to LR prediction method, while using LR to predict the Radio traffic.

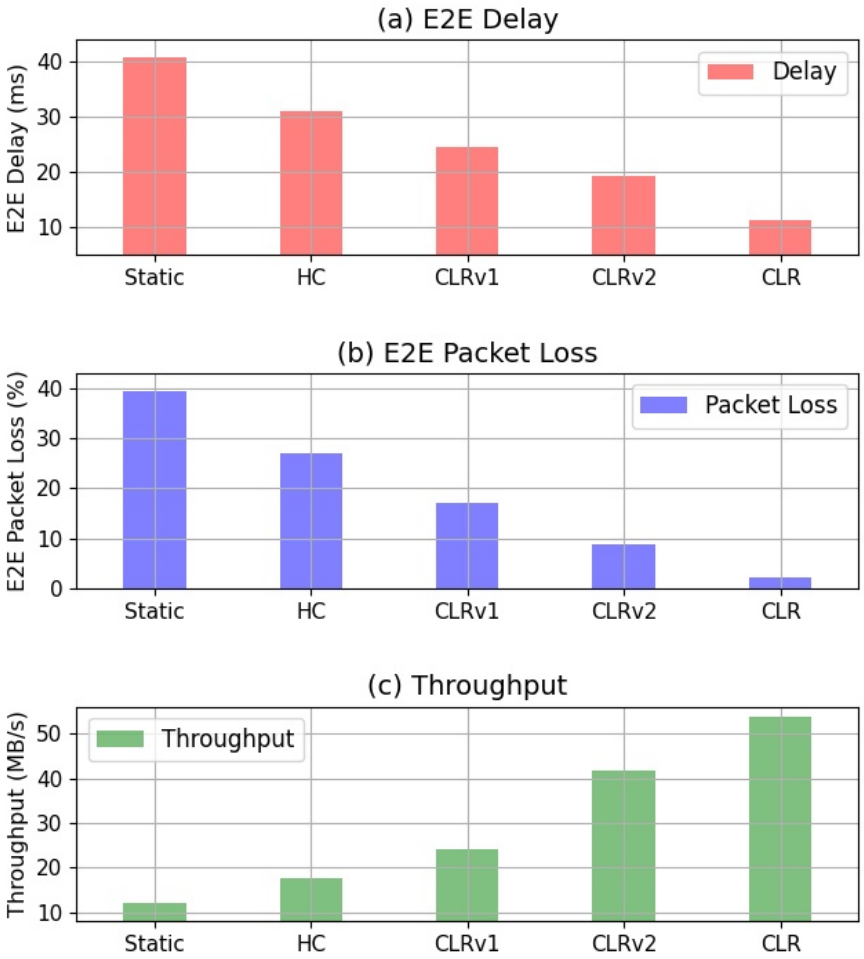

Fig. 5.9 Delay, Packet Loss and Throughput under CLR based rule placement algorithm with and without prediction

Finally, figure 5.10 plots the maximum link utilization of our scheme CLR under two prediction methods (LSTM and LR) compared to HC. We can see that CLR outperforms HC, since some links and devices are overloaded and experiencing congestion while other are under-utilization in the HC approach. However, when using CLR, the traffic load is balanced, which minimizes the link utilization. Moreover, LSTM shows better performances compared to LR thanks to its accuracy in congestion prediction.

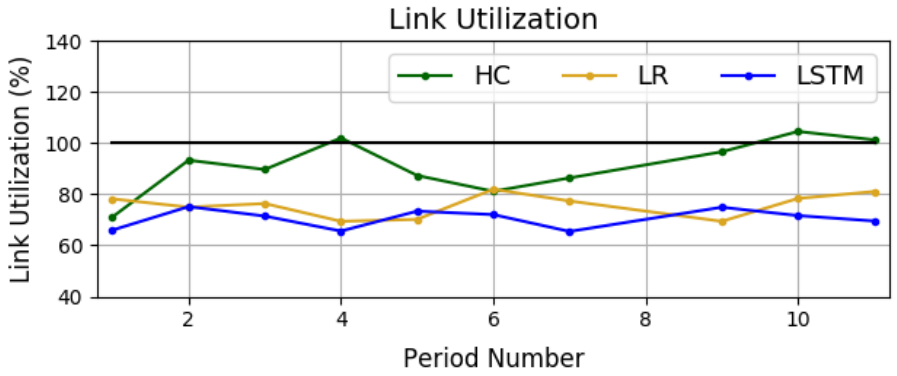

Fig. 5.10 Maximum link utilization under CLR based rule placement algorithm with and without prediction

5.4 Conclusion

In this chapter, we have addressed the design and implementation of a novel architecture based on SDN and ML techniques for enabling creation, modification and continuity of radio and transport network slices, while considering their performances and QoS requirements. To do so, our solution relies on OAI tool, FlexRAN, ONOS and an extended version of OVS that we patched to handle GTP packets. Second, the proposed solution enables efficient sharing of RAN and PN resources by proactively adjusting radio slices capacities and adapt them to the corresponding PN slices, then a balancing of traffic load and congestion prevention have been proposed for improving routing and avoiding congestion within each PN slice. To this end, we have mathematically formulated the problem as a LP which aims to minimize the total network delay and proposed a Congestion prediction and Load balancing Rule (CLR) placement algorithm to solve it with low time complexity and high estimation accuracy. Experimental results show the feasibility of the proposed solution of handling E2E slicing continuity in real-time. Specifically, results show the outperformance of using ML techniques in real-time on other schemes in terms of increasing throughput and deceasing packet loss and latency.

Incorporating intelligence to SDN paradigm through ML can improve considerably the performances and pave the way to address the control plane scalabity. This will be the target of the next chapter.

Chapter 6

Dynamic Clustering of Software Defined Network Switches and Controller placement using Deep Reinforcement Learning

Contents

6.1 Introduction

SDN is an emerging paradigm that allows dynamicity, automation, flexibility and centralized management of the underlying network contrary to traditional networks making it ideal to designing Beyond 5G networks (B5G) that involve essentially higher capacity and lower latency. SDN separates the control plane that is responsible for making routing decisions and the Forwarding Plane, that is only required to perform packet forwarding according to the received routing strategies from the control plane. This communication is enabled via API such as the REST on the northbound interface and OpenFlow on the southbound interface. Even the separation of the control and data planes brings technical benefits, it can cause several drawbacks such as the number of controllers needed, their optimal corresponding placement and which data plane devices to be controlled by which controller.

With the advent of B5G networks, the network size grows exponentially. Therefore, the deployment of a single controller to control the whole network brought greater challenges to the SDN controller's processing capabilities in terms of scalability, performance and reliability. To this end, the use of multiple controllers is primordial, which can be achieved by two main strategies: distributed controllers and replicated controllers. In the replicated controllers strategy, several copies of the SDN controllers have always the same information and keep the full state of the network. Even the fast recovery time when a controller fails, keeping the set of replicated controllers aware of every networking operation, can bring expensive overhead in terms of resource utilisation such as Central Processing Unit (CPU), Random Access Memory (RAM) and storage. On the other hand, the distributed controllers strategy fragments the network into smaller domains, each supervised by a dedicated controller. The set of controllers supervising these domains communicate to each other by their west/eastbound interfaces.

The distributed controllers strategy unveils several challenges that must be considered such as : i) the optimal number of controllers needed for a given network topology, in such a way that, each controller must be not overloaded neither underutilized, ii) the optimization of the controller placement is another concern, as it widely impacts on several fronts such as flow setup latency, resiliency and load balancing and iii) the clustering of the data plane devices in order to improve the intra-cluster performances.

To this end, we propose, in this chapter, a new approach that determines, the optimal number of controllers, their optimal placements as well as the optimal clustering of data plane switches. To do so, we first mathematically formulate the controller placement problem as an optimization one, whose objective is to minimize the controller response time, which corresponds to the delay between the SDN controller and assigned switches, the controller resource utilization, the ICD and the ICT. As the formulated optimization problem is an NP-Hard problem, we first propose to model the problem as a MDP and solve it by a DQN approach. Then, we propose a simple yet efficient heuristic, DDCP that dynamically interacts with the DQN agent to solve the aforementioned optimization problem.

To the best of our knowledge, we are the first to propose and validate such solution using DQN approach. The main contributions of our chapter can be summarized as follows:

- First, we mathematically model the controllers clustering and placement problem as an optimization problem whose objective is to minimize the controller response time, the controller resource utilization, the ICD and the ICT.
- Second, we formalize our problem as MDP with appropriate states and actions, then propose using a DQN approach to solve it.
- Third, we propose simple yet efficient heuristic algorithm called DDCP, that takes as inputs the set of switches and controllers, the trained DQN agent and outputs the optimal clustering in both the control and data planes.

The remainder of this chapter is organized as follows. Section 6.2 discusses the key assumptions and ideas of DDCP. In Section 6.3, we discuss the architecture of the proposed framework, describes the problem formulation and presents our DQN-based heuristic. Section 6.4 evaluates the proposed method. We finally conclude this chapter in Section 6.5.

6.2 DDCP Key Assumptions and Ideas

Different from the approaches proposed in related works, we propose in this chapter the DDCP approach, by exploring the potential of reinforcement learning techniques, such as DQN, for the clustering and placement of controllers, while taking into account four performance metrics: the CD, the CL, the ICD and the ICT. The idea behind using a DQN approach to solve the clustering and controllers' placement problem, is to use only one step (after training) to get the optimal controllers placement as well as the corresponding data plane domains of switches, while considering the four performance metrics. The advantage of using DQN is the ability to improve network performances, while maximizing or minimizing a certain reward. This reward can be modeled as an extensible function, taking into account several parameters (that will be listed in the next section) and can be further extended after training. Moreover, to overcome the problem of scalability and extensibility presented in related works, we implement our DQN agent in the knowledge plane, which is not part of the SDN architecture. This makes our DDCP approach independent of the technology implemented in both the control and data planes.

6.3 DDCP Approach

In this section, we present our approach for the clustering and placement of controllers in SDN using DQN. Firstly, we explain the overall framework. Thereafter, the problem formulation and the Deep Q-Network Agent will be described.

6.3.1 DDCP Architecture

We design our framework according to the KDN paradigm [45], by introducing the knowledge plane to the conventional SDN paradigm, in which we exploit the control plane to have a global view of the network (cf. Fig. 6.1).

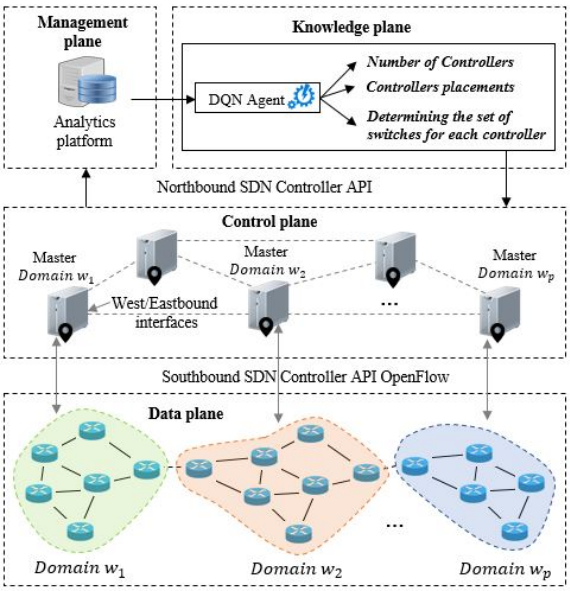

Fig. 6.1 DDCP Architecture

Fig. 6.1 presents our system architecture, which consists of four planes: Data plane, Control Plane, Management Plane and Knowledge Plane.

The data plane that consists of programmable forwarding devices, in charge of data packet processing and forwarding. These devices have no embedded intelligence to take decisions and rely on the control plane to populate their forwarding tables and update their configurations based on the OpenFlow protocol. Moreover, the data plane is divided into multiple domains and each of them is supervised by a dedicated controller.

The control plane is considered as the brain of the SDN network, which incorporates the whole intelligence by abstracting the management and global view of the network in a set of distributed controllers in different locations. Each controller may not have full control or knowledge of the network status and has the responsibility for only a portion of the network (i.e., domain). It communicates with the other controllers through the West/Eastbound interfaces.

The management plane ensures the correct operation and performance of the network by collecting the network measurement from the control plane Network Measurement module, in order to provide network analytic. The collected statistics will be analyzed and sent to the knowledge plane.

In order to not affect the control plane performances, the process of deploying distributed controllers needs to be fully automated and can be ensured by the knowledge plane. This latter exploits the control plane and the management plane by taking the data from the management plane as input

to be fed to ML algorithms, which will convert them to the form of knowledge. Precisely, it learns the behavior of the network, by processing the collected statistics, then determines the number of controllers, their locations and the set of switches embedded in each controller's domain, by deploying a DQN agent. The output of the DQN agent is transmitted to the control plane through the Northbound SDN controller API.

In what follows, we present our controller placement problem formulation, then we detail the Deep Q-Network Agent.

6.3.2 Problem Formulation

We model the SDN network as an undirected graph $G = (V, E)$, where $V = \{v_j\}$ is the set of switches and $|V| = k$ is the number of switches and E is the set of edges (i.e., links between switches). The control plane consists of a set of controllers $C = \{c_i\}$, where $|C| = n$ denotes the number of controllers. On the other hand, the data plane is fragmented into a set of domains $W = \{w_l\}$, each domain $w_l = \{v_j\}, v_j \in V$ supervised by a controller, and $|W| = p$ denotes the number of domains. The solution of our Controller Placement Problem (CPP) can be represented as a binary vector $F = (F_1, F_2, ..., F_n) \in R^n$, where F_i is given by:

$$F_i = \begin{cases} 1, & \text{if controller } c_i \text{ is selected} \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

Since our objective is to optimize the number of deployed controllers, while guaranteeing the QoS requirements of all traffic requests in the network, we define here-after four different variables to compute the number of selected controllers: Control Load (CL), Control Delay (CD), Intra-Cluster Delay (ICD), and Intra-Cluster Throughput (ICT).

Number of Selected Controllers

To determine the number of deployed controllers p (i.e., the number of data plane domains), we first, define the variable R^t_{ij} representing the total flow request from switch (j) to controller (i) at time (t), which corresponds to the number of *Packet-In* messages generated by the switch. It is worth noting that, the *Packet-In* messages are generated and sent from switches to the controller when there is no matching flow entries in their flow tables. Secondly, we assume that the portion of resources consumed by one flow request concerns essentially the CPU and RAM, and can be written as follows:

$$\lambda_i = \frac{CPU_{flow_i}}{CPU_i} + \frac{RAM_{flow_i}}{RAM_i} \tag{6.2}$$

Where CPU_i and RAM_i correspond, respectively, to the maximum capacity of CPU and RAM of the controller c_i. As CPU and RAM use different units, we divide them over their corresponding maximum values to get normalized data. Therefore, the number of selected controllers can be written as follows:

$$p = \sum_{i=1}^{n} \sum_{j=1}^{k} R_{i,j}^{t}.\lambda_i \tag{6.3}$$

Control Load (CL)

We define the CL as the load incurred by all the switches belonging to the same domain. To this end, we use a decision variable $H_{i,j}^{t}$ to determine the relationship between the controller (i) and the switch (j) as follows:

$$H_{i,j}^{t} = \begin{cases} 1, & \text{if switch } v_j \text{ is controlled by } c_i \text{ at time } t \\ 0, & \text{otherwise} \end{cases} \tag{6.4}$$

The load of the controller c_i can be thus given by:

$$CL_i^t = \sum_{j=1}^{k} H_{i,j}^{t}.R_{i,j}^{t}.\lambda_i \tag{6.5}$$

To avoid the overloading of some controllers and the under-utilization of others, we balance the traffic load between the set of selected p controllers. To this end, we determine the global CL as follows:

$$CL^t = \frac{1}{p} \sum_{i=1}^{p} |CL_i^t - CL_{avg}^t| \tag{6.6}$$

Where CL_{avg}^t denotes the average of the CL of all selected p controllers.

Control Delay (CD)

The CD corresponds to the average response time of the controller c_i, which is defined as follows:

$$CD_i^t = PD_i^t + 2.CMD_i^t \tag{6.7}$$

Where PD_i^t and CMD_i^t are, respectively, the processing delay and the communication delay of the controller c_i at time t. We used two times of the communication delay since the *Packet-In* comes from the switch to the controller and returns back to the switch. In this way, the global CD is determined as the average of the CD of the set of selected controllers, which can be written as follows:

$$CD^t = \frac{1}{p} \sum_{i=1}^{p} CD_i^t \tag{6.8}$$

According to [95], the processing delay is determined as follows:

$$PD_i^t = \frac{1}{\varphi_i - \theta_i} \tag{6.9}$$

Where φ_i and θ_i are, respectively, the capacity and the workload of the controller c_i. The communication delay is determined as follows:

$$q_i^t = \sum_{j=1}^{k} H_{i,j}^t \tag{6.10}$$

$$CMD_i^t = \sum_{j=1}^{k} \frac{H_{i,j}^t . d_{i,j}^t}{q_i^t} \tag{6.11}$$

Where q_i^t denotes the number of switches supervised by the controller c_i and $d_{i,j}^t$ denotes the delay between the controller c_i and the switch v_j. The latter is measured based on our previous work in [34].

Intra-Cluster Delay (ICD)

This metric corresponds to the average value of the propagation delays between all the switches belonging to the same cluster i, which can be written as follows:

$$ICD_i^t = \sum_{v_e \in w_i, v_m \in w_i} \frac{A(v_e, v_m).\psi(v_e, v_m)}{\varsigma_i} \tag{6.12}$$

Where $A(v_e, v_m)$ denotes the delay between nodes v_e and v_m in the domain w_i, ς_i is the number of links of the cluster i, and $\psi(v_e, v_m)$ is a decision variable representing the relationship between any two switches, which is defined as follows:

$$\psi(v_e, v_m) = \begin{cases} 1, & \text{if } v_e \text{ is connected to } v_m \\ 0, & \text{otherwise} \end{cases} \tag{6.13}$$

Then, the global ICD of the set of p clusters is determined as follows:

$$ICD^t = \frac{1}{p} \sum_{i=1}^{p} ICD_i^t \tag{6.14}$$

Intra-Cluster Throughput (ICT)

This metric corresponds to how much data can be transferred by a specific data plane cluster (i) within a given timeframe, which referred to us ICT_i^t. Then, the global ICT of the set of p clusters is determined as follows:

$$ICT^t = \frac{1}{p} \sum_{i=1}^{p} ICT_i^t \tag{6.15}$$

6.3.3 Controller Placement and Switches Migration

We consider the controllers as images installed in different servers located in different locations. In this way, the action of controllers placement refers to instantiating a container from the image in

the corresponding server located in a specific location. To this end, we consider $c_i, i \in [1,n]$ as the instance of the controller in the server or location i. Also, the deselection follows the same logic by just deleting the instance or container c_i.

It is worth noting that the clustering of control and data planes as well as the placement of the controllers happen when certain controllers are overloaded, while others are under-utilized. Hence, we define in equation (6.16) the load balancing factor between clusters, calculated as follows:

$$CL_{mig} = \frac{CL^t}{max(CL_i)} \times 100 \tag{6.16}$$

Note that, finding new data plane clusters leads to migrating a set of switches from old clusters controlled by specific controllers to new clusters controlled by new controllers.

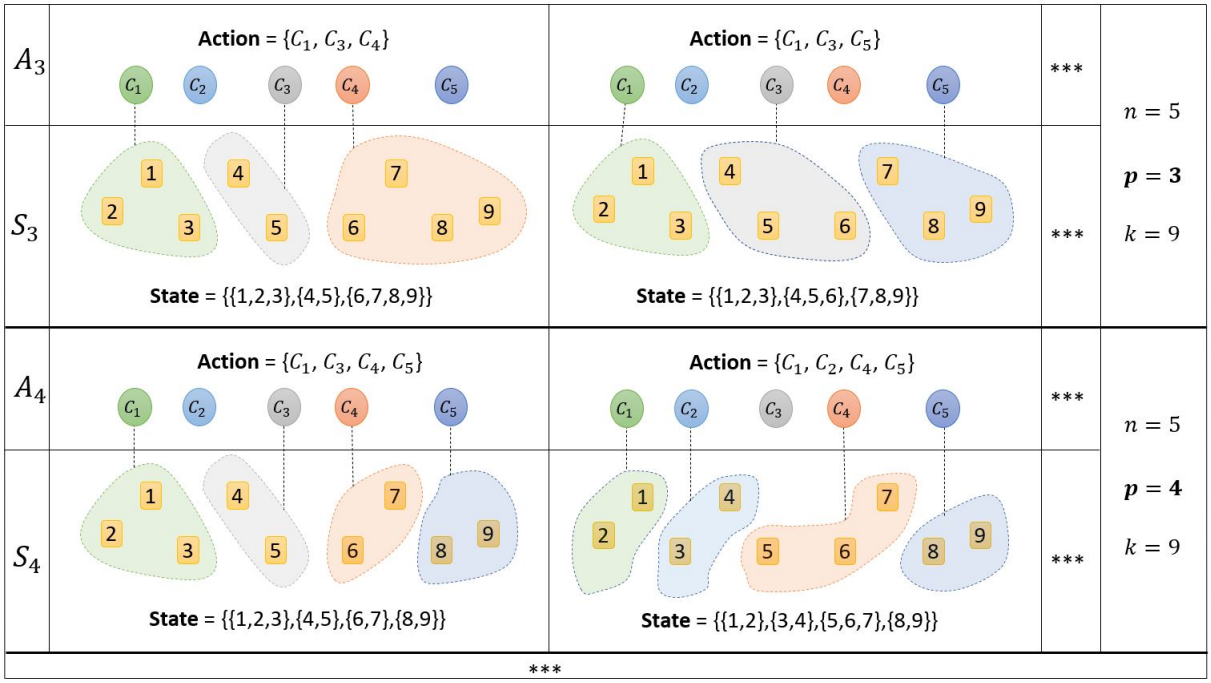

Fig. 6.2 Demonstration of the DQN State and Action spaces using a simple topology

6.3.4 Proposed Optimization Model

The objective of our optimization model is to minimize the aforementioned performance metrics including CL, CD, ICD and ICT. This can be achieved as follows:

$$Min(\alpha \times \frac{CD^t}{CD_{max}} + \beta \times \frac{CL^t}{CL_{max}} + \gamma \times \frac{ICD^t}{ICD_{max}} - \rho \times \frac{ICT^t}{ICT_{max}}) \tag{6.17}$$

Subject to :

$$\forall\, i \in [1,p] : CL_i^t < M.F_i \tag{6.18}$$

$$\forall\ (w_u, w_v) \in W^2 : w_u \cap w_v = \oslash \tag{6.19}$$

$$\text{if controllers } c_i \text{ and } c_j \text{ are selected: } c_i \neq c_j \tag{6.20}$$

$$|W| = p \tag{6.21}$$

$$\forall\ (i,l) \in [1,p]^2, \forall\ j \in [1,k] : (d_{i,j} < d_{l,j}) \Rightarrow (H^t_{i,j} > H^t_{l,j}) \tag{6.22}$$

$$\forall\ j \in [1,k] : H^t_{i,j} \leq F_i \tag{6.23}$$

$$\forall\ i \in [1,p] : (\sum_{j=1}^{k} H^t_{i,j} = 0) \Rightarrow (F_i = 0) \tag{6.24}$$

$$\forall\ i \in [1,p] : CD^t_i \leq \sigma_i \tag{6.25}$$

$$\forall\ (v_e, v_m) \in V^2 : A(v_e, v_m) \leq \delta \tag{6.26}$$

As the objective function involves different parameters with different measurement units, we have divided each metric over its corresponding maximum value to have a normalized objective function. Note that these maximum values are determined by the network operator and correspond to physical characteristics of involved network devices. Note also that α, β, γ, and ρ are adjustable weighting factors determining the degree of importance of the CD, the CL, the ICD and the ICT metrics, respectively, such that $\alpha + \beta + \gamma + \rho = 1$.

Constraint (6.18) forces all controllers to not be overloaded. Constraint (6.19) means that all domains do not overlap and each node belongs to only one domain. Constraint (6.20) forces the set of controllers to be selected only once. Constraint (6.21) insures that the number of selected controllers is the same as the number of data plane domains. Constraint (6.22) is the mapping constraint ensuring that a switch must be mapped to the closest controller in terms of delay. Recall that, we refer to our work in [35] to measure the delay between the switches in the data plane as well as between the switches and their corresponding controllers, where the delay is measured based on the times of sending and receiving a specific packet probe from the controller to the switches in the data plane. Constraint (6.23) means that a switch is mapped to a controller if the latter exists and is selected. Constraint (6.24) means that if there is no switch mapped to a controller then the latter will be powered off. Constraint (6.25) forces the CD metric to not exceed a certain threshold σ_i, and finally constraint (6.26) forces the links of each cluster to not be delayed.

The formulated optimization problem is an NP-Hard problem, where the optimal solution is very difficult to obtain in general. To this end, we propose to solve it by modeling and training a Deep Q-Network Agent, as will be detailed in the next section.

6.3.5 Deep Q-Network Agent

To dynamically determine the optimal number of controllers and their optimal placements while considering the propagation delay between the switches and the controller, the controller resource

utilization, the ICD and the ICT, we propose to model a DQN Agent based on a MDP. In this way, the DQN agent interacts with the environment through three signals: $State$,$Action$ and $Reward$.

State

The State $S_{t,p}$ corresponds to the partitioning of the data plane into p domains. To do so, we define it as a vector of $w_{t,i}$ ($i \in [1,p]$), and can be written as follows:

$$S_{t,p} = [w_{t,1}, w_{t,2}, ..., w_{t,p}]$$

Where $w_{t,i} = [v_e, ..., v_m], \forall (v_e, v_m) \in w^2_{t,i}$ is a vector representing a cluster of switches, such that $\sum_{i=1}^{p} |w_{t,i}| = k$. Recall that, k corresponds to the total number of switches in the data plane.

Let S_p denote the set of all states corresponding to a specific value of p, written as follows:

$$S_p = [S_{1,p}, S_{2,p}, ..., S_{T_p,p}]$$

Where T_p corresponds to the number of states corresponding to p clusters. It is worth noting that, the set of states are constructed by respecting the set of constraints indicated in Section 6.3.4.

Action

The action taken by the agent $A_{r,p}$ is characterized by a vector representing a selected set of p controllers from the available n controllers, which is defined as follows:

$$A_{r,p} = [c_{r,1}, ..., c_{r,p}], \forall c_{r,i} \in C$$

Where r represents the action number. We denote the set of all actions corresponding to a specific value of p as follows:

$$A_p = [A_{1,p}, A_{2,p}, ..., A_{R_p,p}]$$

Where R_p corresponds to the number of actions corresponding to p clusters. It is worth noting that, the set of actions are constructed by respecting the set of constraints (6.20) and (6.21) indicated in Section 6.3.4. Recall that, constraint (6.20) avoids selecting one controller for more that one cluster and constraint (6.21) ensures that the number of controllers is the same as the number of data plane domains.

Reward

The "Reward" function R of the agent consists in minimizing the normalized objective function defined in (6.17), and can be thus written as follows:

$$R = \alpha \times \frac{CD}{CD_{max}} + \beta \times \frac{CL}{CL_{max}} + \gamma \times \frac{ICD}{ICD_{max}} - \rho \times \frac{ICT}{ICT_{max}} \tag{6.27}$$

It is worth noting that the proposed DQN agent consists in determining the best mapping between the set of states and the set of actions, while maximizing $1/R$ (i.e., minimizing R).

In order to give more detail on the DQN design, we propose to illustrate it graphically in a small topology with 9 switches in the data plane and 5 controllers in the control plane, as shown in Fig. 6.2. We can see that, when $p = 3$ the data plane is fragmented into three domains as well as only three controllers are used to supervise each domain and the rest of controllers are not instantiated. In this case, the state corresponds to the selected three domains and the action corresponds to the selected three controllers. When $p = 4$, we can see that the shapes of the state and action vectors are changed, where four controllers from five are selected for each action and the data plane is fragmented into four domains.

Recall that, the number of controllers to be selected, corresponds to the number of data plane domains or clusters, and can take values from 1 to n, where n corresponds to the total number of controllers in the control plane.

6.3.6 DDCP Heuristic

As the state $S_{t,p}$ and action $A_{r,p}$ take different forms or shapes for each value of p, we propose to split the state space (i.e., training data) based on p values. Then, we train the DQN agent separately for each value of p (i.e., DQN($S_{t,p}$, $A_{r,p}$)). In this way, to determine the optimal number of controllers p and the best mapping between the set of states (clusters of switches) and the set of p controllers, we propose the DDCP heuristic (i.e., Algo. 3):

The DDCP algorithm consists of two main procedures: *Clust* and *Migration*. The *Clust* procedure is called to determine the set of controllers and their corresponding placements as well as the set of data plane clusters. It works as follows: it takes as input the set of splitted states and the set of splitted actions based on p: $S = \{S_1, S_2, ..., S_n\}, A = \{A_1, A_2, ..., A_n\}$. Recall that, S_p corresponds to the space of states where the number of controllers as well as the number of data plane domains is p. Then, by using the trained DQN Agent, it finds the optimal *State*, *Action* corresponding to the maximum reward (lines 4-15) iteratively for each sub-states $S_p \in S, p \in [1, n]$. The output of the *Clust* procedure is the optimal number of controllers to be deployed (*Select_p*), their corresponding ID ($Select\_Act \in A_{Select\_p}$), and the corresponding set of switch clusters ($Select\_St \in S_{Select\_p}$) (line 16).

On the other hand, the *Migration* procedure is called when the control plane load is not well-balanced (lines 19). In this case, it takes as input the output of the *Clust* procedure (i.e., *Select_p*, *Select_Act*, *Select_St*). Then, it migrates the set of switches to the new cluster if the new controller is different from the old one (lines 20-31).

Note that, the *Map* function indicates if a switch in *Select_St* is mapped to a controller in *Select_Act*. The *CTRL_old* represents the controller of a specific switch before migration.

Algorithm 3: DDCP algorithm

procedure CLUST($S = \{S_1, ..., S_n\}, A = \{A_1, ..., A_n\}$)
    $Max\_Reward \leftarrow 0$
    $p \leftarrow 1$
    while $p \leq n$ **do**
        for *each* $state \in S_p$ **do**
            $Reward,\ Action \leftarrow DQN(state, S_p, A_p)$
            if $Reward > Max\_Reward$ **then**
                $Max\_Reward \leftarrow Reward$
                $Select\_St \leftarrow State$
                $Select\_Act \leftarrow Action$
                $Select\_p \leftarrow p$
            end if
        end for
        $p \leftarrow p + 1$
    end while
    return $Select\_St, Select\_Act, Select\_p$
end procedure
procedure MIGRATION($Select\_St, Select\_Act$)
    if $CL_{mig} > Threshold$ **then**
        for *each* $CTRL \in Select\_Act$ **do**
            for *each* $Domain \in Select\_St$ **do**
                if $Map(CTRL, Domain)$ **then**
                    for *each* $Switch : s \in Domain$ **do**
                        if $CTRL\_old(s) \neq CTRL$ **then**
                            $Remove(s, CTRL\_old)$
                            $Assign(s, CTRL)$
                        end if
                    end for
                end if
            end for
        end for
    end if
end procedure

6.4 Performance Evaluation

In this section, we evaluate the efficiency of our proposed approach. We start by presenting our experimental setup. Then, we present the experimental results.

6.4.1 Experimental Setup

First, the control plane is deployed as a cluster of a set of dockerized OpenFlow ONOS [26] controllers. Then, the DQN agent is implemented based on Python [7] and dockerized on Docker Containers [52].

Table 6.1 DQN parameters for DDCP

Name	Value
Dense layers	2
Control plane capacity	10
Data plane capacity	32
Minimum number of switches per cluster	1
Maximum number of switches per cluster	13
Q-target network update frequency	200
Learning rate	0.01
Discounted factor	0.6
Mini-batch size	32
Final exploration rate	0.2
Memory size	2000 units
Number of episodes	1000

The latter interacts with the control plane based on the ONOS Northbound API. The control plane consists of 12 controllers. We used the network emulation tool OpenvSwitch [120] to implement the experimental topology, which consists of 32 nodes. To generate traffic among hosts, we used Iperf [77]. The control plane Network Measurement modules collect statistics (latency, throughput, and per-flow size) from the devices and report those time-series statistics to the InfluxDb database [3]. Note that, to show the importance of using DQN on a large space of states and actions, we augmented the collected statistics by using data generated with Python.

Recall that, the DQN model consists of two neural networks, designed to improve the convergence of the Cost Function in (2.19). One neural network is called *Q-Network* to estimate the Q-Values and the second one is called the *Q-target* to estimate the target network, according to the mechanism shown in Fig. 2.11.

We built and trained the DQN model by using the Tensorflow library [9], by deploying separately the two neural networks (i.e., *Q-Network*, *Q-target*), which have the same architecture. The DQN parameters are illustrated in Table 6.1. In particular, both the *Q-Network* and the *Q-target* consist of 2 dense layers. The number of data plane switches is 32 and the number of controllers in the control plane is 10.

Considering the implemented topology, and in order to not have a huge state space size, the training data are built as follows: i) the minimum and maximum number of switches in each cluster are fixed to 1 and 13, respectively, ii) the clusters where no link between the generated switches of a specific cluster are ignored. On the other hand, the training data (i.e., the set of States) can be classified based on p, as the State corresponds to a vector of p sub-vectors. In this way, both the set of States and the set of Actions are splitted based on p, as illustrated in Fig. 6.3. Then, we trained separately a set of DQN agents according to p. Recall that, we denoted the number of states corresponding to p by T_p and the number of actions corresponding to p by R_p in Sections 6.3.5 and 6.3.5, respectively. This

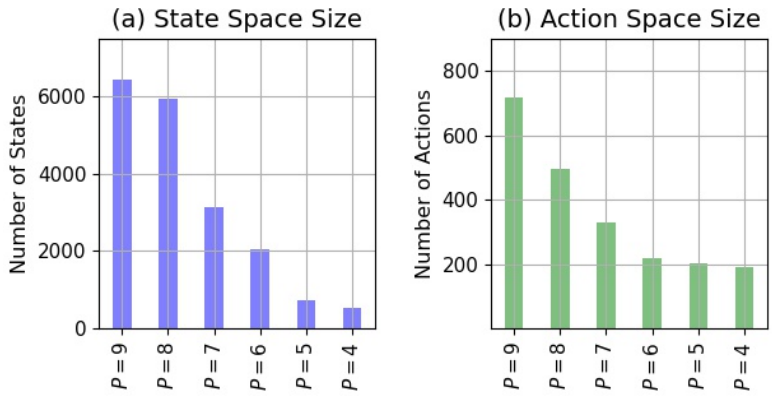

Fig. 6.3 Number of States and Actions under different number of clusters p

mechanism of splitting the training data helps our proposed DDCP approach to determine the best clustering of control and data planes, as we need to go through the set of all trained DQN agents.

It is worth noting that, the global number of States and Actions is selected based on a set of parameters: i) the global number of switches in the data plane (i.e., 32 switches in our case) and the number of controllers in the control plane (i.e., 10 controllers in our case), ii) the maximum and minimum capacity of each cluster, and iii) the convergence of the Cost function, while training the set of DQN agents based on p.

During the training phase, we adopt ε-greedy method as action selection method. The final exploration rate is fixed at 0.2, while the *Q-target* parameters are copied from the *Q-Network* every 200 steps. The learning rate and discounted factor are fixed to 0.01 and 0.6, respectively, which correspond to η and φ parameters in equations (2.17) and (2.18). In addition, each training process corresponds to 1000 episodes. Finally, we fixed the $Threshold$, indicated in Algorithm 3, to 30% to avoid overloading the control plane, as will be justified in the next section.

6.4.2 Experimental Results

In order to evaluate the performance of our proposed DDCP approach, we first determine the best DQN model based on the reward function weighting factors. Then, we determine the number of controllers (i.e., clusters) p to be deployed. To do so, we compare our approach with the well-known K-means clustering method. Then, we show the benefit of our approach in term of data plane partitioning i.e., which switch assigned to which cluster. Thereafter, we evaluate its performances in terms of CD, CL, ICD and ICT metrics. Finally, we perform a comparative analysis between our proposed DDCP approach and three main schemes proposed in the literature: 1) Optimal and Dynamic Controller Placement (ODCP) [109], 2) Dynamic SDN Controller Placement in Elastic Optical Datacenter Networks (DSCP) [97], and 3) A Hierarchical K-means Algorithm for Controller Placement in SDN-based WAN Architecture (HKCP) [93].

As mentioned in equation (6.27), the reward function is composed of four performance metrics (i.e., CD (CD_i^t), CL (CL_i^t), ICD (ICD_i^t) and ICT (ICT_i^t)) weighted by four parameters α, β, γ and ρ, respectively. These weighting factors play an important role to determine, in one side, the importance of each performance metric, and on the other side the convergence of the Loss function, shown in equation (2.19). To this end, we depict, in Fig. 6.4, the average value of each performance metric (i.e., CD, CL, ICD, ICT) after training the DQN agent under different number of training episodes (1000 episodes in total), while changing the weighting factors according to the following strategies:

- S_1: this strategy considers only the control plane performance metrics (i.e., CD and CL): $\alpha = \frac{1}{2},\ \beta = \frac{1}{2},\ \gamma = 0,\ \rho = 0$.
- S_2: this strategy considers only the data plane performance metrics (i.e., ICD and ICT): $\alpha = 0,\ \beta = 0,\ \gamma = \frac{1}{2},\ \rho = \frac{1}{2}$.
- S_3: this strategy considers delay-throughput performance metrics (i.e., CD, ICD, ICT): $\alpha = \frac{1}{3},\ \beta = 0,\ \gamma = \frac{1}{3},\ \rho = \frac{1}{3}$.
- S_4: this strategy gives importance to all performance metrics: $\alpha = \frac{1}{4},\ \beta = \frac{1}{4},\ \gamma = \frac{1}{4},\ \rho = \frac{1}{4}$.

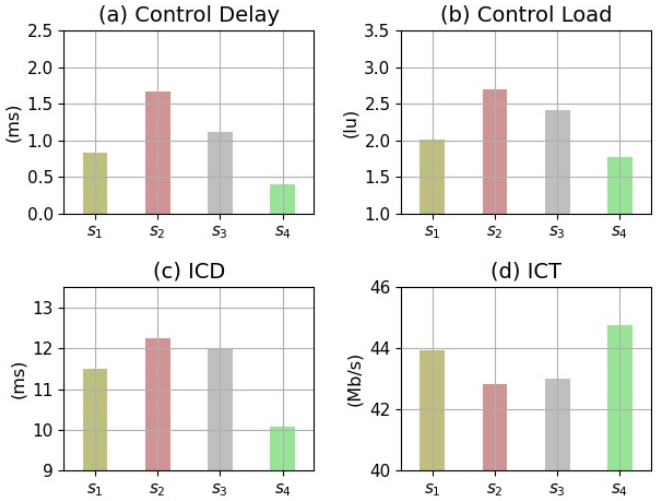

Fig. 6.4 Impact of varying the reward function weighting factors $\alpha, \beta, \gamma, \rho$ on the CD, the CL, the ICD and the ICT metrics

From Fig. 6.4, we can see that considering only the data plane performance metrics in strategy S_2 causes obviously high values of CD, CL and ICD, while decreasing the ICT metric. The reason is that some controllers are overloaded and experiencing congestion, while others are under-utilized. On the other hand, strategy S_3 shows better performances compared to strategy S_2, since the CD metric is taken into account. However, the control load (CL) is still high compared to the two remaining strategies (S_1 and S_4) since this metric is not taken into account in the reward function of strategy S_3.

Considering the CL metric in strategy S_1 improves the performances compared to the two strategies S_2 and S_3. This shows the importance of balancing the load between the set of controllers in the control plane. Finally, strategy S_4, which takes into account all performance metrics (i.e., CD, CL, ICD and ICT), outperforms all others strategies, showing high throughput, low intra-cluster delay, low control delay, and low control load. Hence, according to these results, we adopt strategy S_4 (i.e., $\alpha = \frac{1}{4}$, $\beta = \frac{1}{4}$, $\gamma = \frac{1}{4}$, $\rho = \frac{1}{4}$) for our subsequent experiments.

Let us now determine the optimal number/range of controllers (i.e., clusters) to be deployed p. To do so, we plot in Fig. 6.5 the evolution of the reward and loss functions (as defined in equations (6.27) and (2.19)) under different number of training episodes and using the aforementioned weighting factors by adopting the strategy S_4. We compare in Fig. 6.5(a) the following baselines:

- $R_4, R_5, R_6, R_7, R_8, R_9$: where R_p corresponds to the reward under the number of clusters p, $p \in [4..9]$.

Similarly, we compare in Fig. 6.5(b) the following baselines:

- $C_4, C_5, C_6, C_7, C_8, C_9$: where C_p corresponds to the cost under the number of clusters p, $p \in [4..9]$.

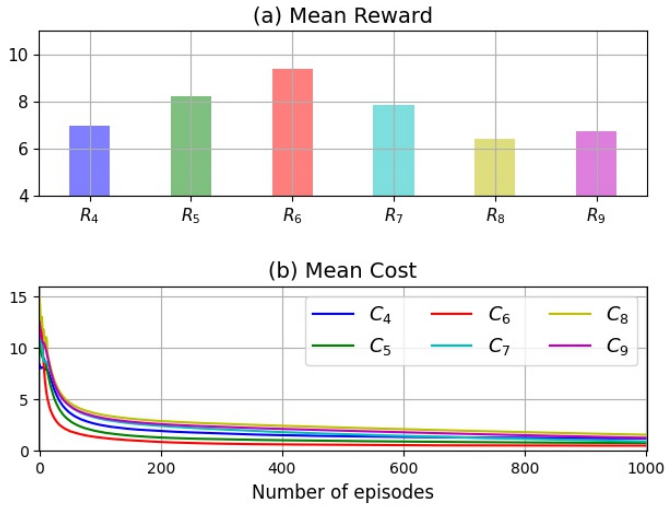

Fig. 6.5 Impact of varying the number of clusters and the controllers placement while training the DQN agent

From Fig. 6.5 (b), we can see that the Loss function for all studied baselines converges. On the other hand, we can observe from Fig. 6.5 (a) that the mean reward increases while increasing the number of clusters in the fist part of the range where $p \in [4..6]$. However, it starts to decrease in the second part of the range where $p \in [7..9]$. The increase in the first part reflects the existence of new clustering configurations (i.e., based on link latency) and controllers placement that lead to minimize the CD, the CL and the ICD metrics. The decreasing in the second part of the range of p, can be

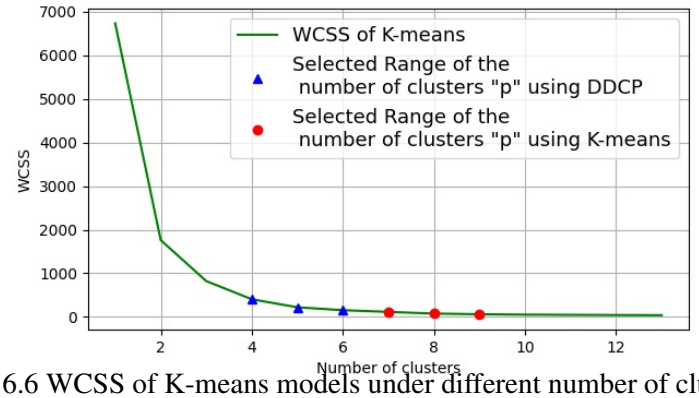

Fig. 6.6 WCSS of K-means models under different number of clusters

explained by the fact that the increase in the number of controllers in the control plane impacts the performances such as the CL metric. As a result, the number of controllers or clusters to be deployed, according to our DDCP approach, corresponds to that of the maximum reward, which is equal to 6 in our experiments. Next, for the sake of comparison, we take this range $[4..6]$ for the variable p.

Let us now see the returned number of controllers p to be deployed when using the well-known K-means clustering method. Recall that K-means is widely used in network partition problems [65] and includes four main steps: 1) select p random points as cluster centers called $Centroids$, 2) assign switches to the closest cluster based on the latency of links and their locations, 3) recalculate the centroid for each cluster by computing the average of the assigned switches, 4) repeat steps 2 and 3 until none of the cluster assignments change.

To determine the number of controllers to be deployed using the K-means algorithm, we refer to the Within Cluster Sum of Squares (WCSS) method [136], which computes the distance (i.e, delay in our case) between a centroid of a cluster and each observation (i.e., switch in our case) based on which it assigns the observation to the nearest cluster. To do so, we plot, in Fig. 6.6, the WCSS method after training the K-means algorithm for maximum 300 iterations under different number of clusters of switches, where WCSS is determined as follows:

$$WCSS = \sum_{i=1}^{p} \sum_{j=1}^{|w_i|} (x_j - y_i)^2 \tag{6.28}$$

$$x_j = (v_{src}, v_{dst}, A(v_{src}, v_{dst})), (v_{src}, v_{dst}) \in w_i^2 \tag{6.29}$$

Where x_j represents a link in the domain w_i, in which v_{src}, v_{dst} are data plane switches and $A(v_{src}, v_{dst})$ is the delay between them, the y_i denotes the centroid of the domain w_i. In this way, the WCSS method consists in clustering the set of points x_j in order to minimize the latency between switches.

From Fig. 6.6, we can see that the average controller response time of WCSS continues to decrease, while increasing the number of clusters, since the more we increase the number of clusters, the more we have a small number of switches in each cluster w_i, which leads to minimizing the latency.

However, this can lead to overload the control plane due to the increase of the number of controllers. To this end, we select the number of clusters (i.e., controllers) when the WCSS starts to have low values and be stable. Moreover, for the sake of comparison using our approach (i.e., DDCP), we select a range of p values when the WCSS converges. This leads us to choose the range $[7,8,9]$ for the number of clusters p based on the K-means algorithm. This results in higher deployed clusters compared to our DDCP approach, where the identified range of p is $[4..6]$.

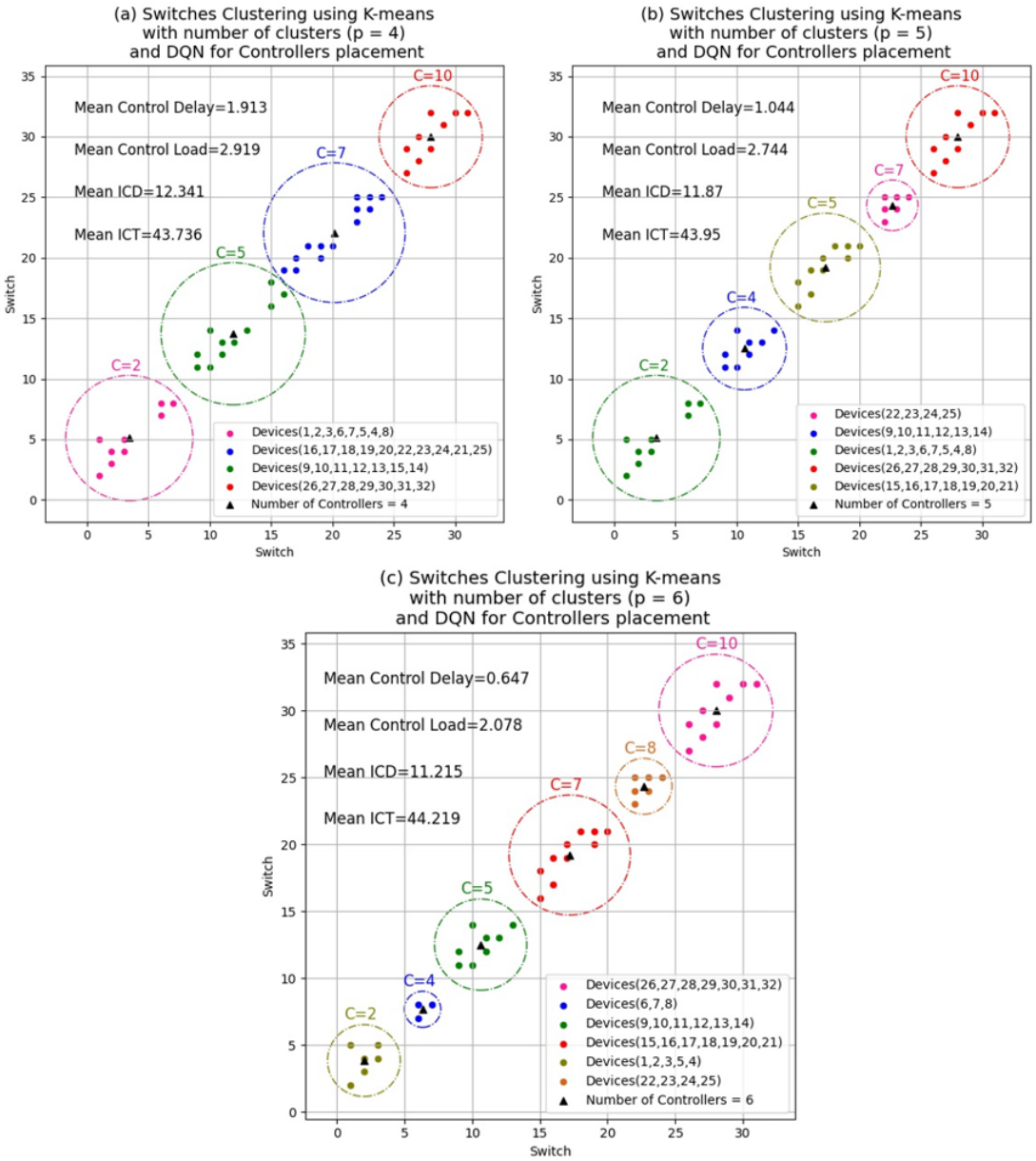

Fig. 6.7 Demonstration of the network clustering for the Reduced DDCP scheme

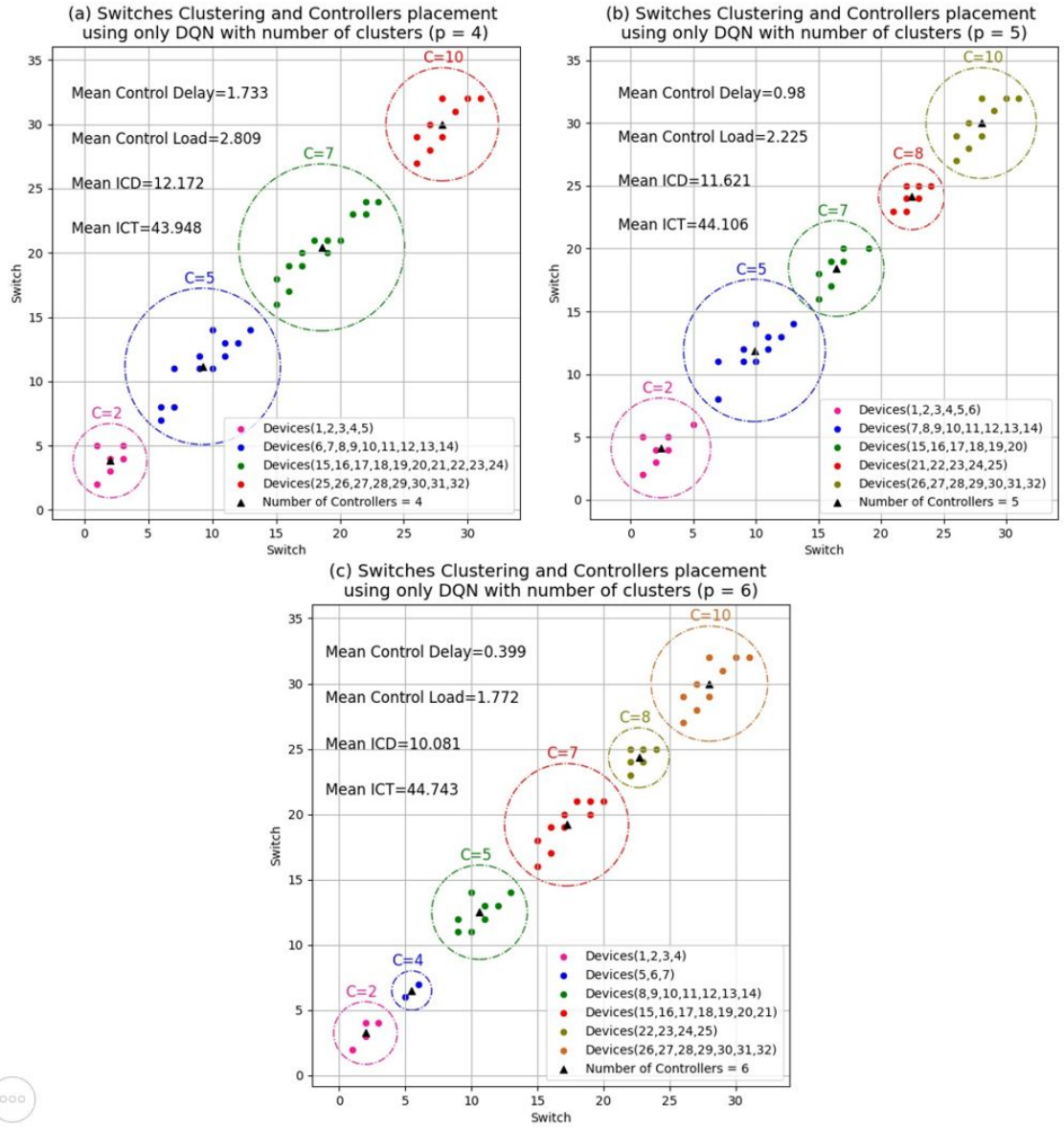

Fig. 6.8 Demonstration of the network clustering for the DDCP scheme

To further show the benefit of our DDCP approach in term of data plane partitioning (i.e., which switch is assigned to which cluster), we first compare it with the following baseline:

- Reduced DDCP, which corresponds to our proposed approach in which the data plane clustering is determined by the K-means algorithm instead of the DQN agent, and where the number of controllers to be deployed is $p \in [4..6]$ (for the sake of comparison) and identified statically.

Fig. 6.7 and Fig. 6.8 depict the data plane devices allocated to each cluster by using the Reduced DDCP and DDCP schemes, respectively, under different values of p. It is noteworthy that, we have used the p values in the range $[4,5,6]$. Also, the reward function parameters of the DQN agent

correspond to the strategy S_4 (i.e., $\alpha = \frac{1}{4}$, $\beta = \frac{1}{4}$, $\gamma = \frac{1}{4}$, $\rho = \frac{1}{4}$). We can see that, when the number of clusters p is equal to 4, the allocation of switches following the two schemes (i.e., Original DDCP and Reduced DDCP) in Fig. 6.7(a) and 6.8(a) is completely different, since the DDCP scheme is based on DQN that uses the previous clustering experiences, while the K-means method used in the Reduced DDCP scheme, is based on recalculating the centroid of each cluster for each step of the model training. When $p = 5$ (cf. Fig. 6.7(b) and Fig. 6.8(b)), we can observe more similarity, compared to those when $p = 4$, mostly in the last cluster. When $p = 6$ (cf. Fig. 6.7(c) and Fig. 6.8(c)) the two schemes achieve interestingly a close clustering result with a superiority of the DDCP approach in terms of average CD, CL, ICD and ICT metrics, as clearly depicted in Fig. 6.9. This convergence in clustering can be explained by the fact that, as the K-means considers only the clustering in the data plane, referring to the DDCP approach to determine the number of clusters improves the performances. However, it still causes some degradation comparing to the DDCP approach since the controllers are identified statically in the reduced one, which increase the CD, the CL and the ICD metrics, as clearly depicted in Fig. 6.9.

Finally, we perform, in the following, a comparative analysis between our proposed DDCP approach and three main approaches proposed in the literature:

- ODCP [109], which consists in using a quadratic program to solve the controller placement problem and determines the number of switches in each switch domain. After solving the controller placement problem, it dynamically migrates switches in case of controller overload.
- DSCP [97], which is based on dynamically matching the set of controllers to the set of data plane switches in order to maximize the resource utilization. Moreover, it dynamically balances the traffic load and deploys the controllers.
- HKCP [93], which is a SDN network partitioning method based on the hierarchical K-means algorithm. It considers the latency between the switches and their controllers as well as the load balancing between the set of controllers.

We used the same experimental topology, described in Section 6.4.1, for all approaches. In addition, to have a fair comparison, we have compared all approaches before and after switch migration. Recall that the action that triggers the switch migration is the overloading of the control plane. To this end, we used the following scenario:

We consider the set of controllers and data plane clusters obtained by using our DDCP approach from Fig. 6.8, where the number of clusters is 5 before switch migration. Then, we force the switches to overload the set of controllers by sending a high number of *Packet_In*. This will force the four studied approaches (DDCP, ODCP, DSCP, and HKCP) to perform the migration of some switches to new clusters and change the network topology.

First, it is interesting to see the impact of varying the Threshold defined in our DDCP approach in the switch migration procedure. To do so, we vary this parameter, denoted by Th, between 10%

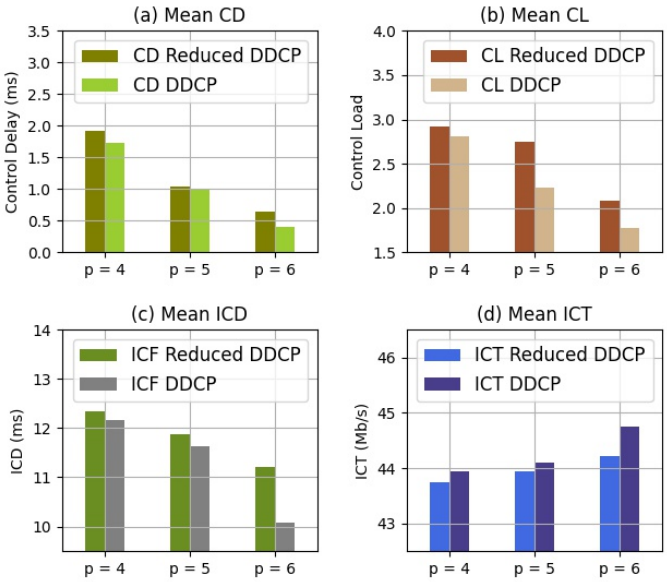

Fig. 6.9 Comparing dynamic clustering and placement based on DQN and K-means methods

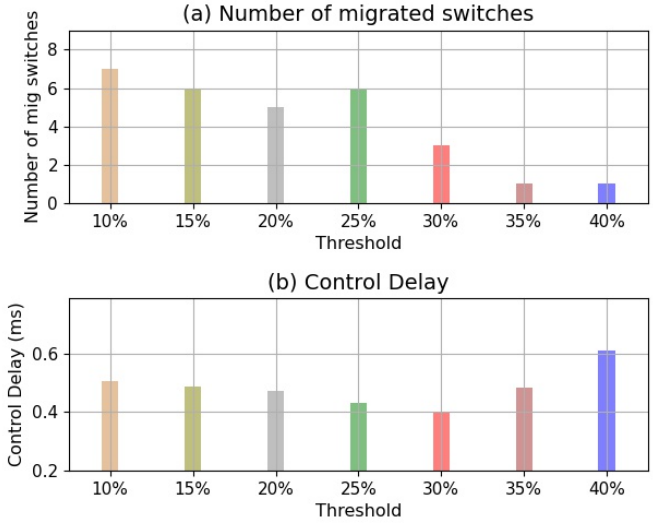

Fig. 6.10 Impact of varying the Threshold (Th) on the number of migrated switches and control delay in the DDCP approach

and 40% and depict the number of migrated switches in Fig. 6.10(a). Recall that CL_{mig}, defined in equation (6.16) and used in the switch migration procedure, refers to the load balancing factor between clusters. The more this factor is high, the more some controllers are overloaded compared to the others. In this way, when CL_{mig} exceeds the Threshold (Th), the migration process is triggered. From Fig. 6.10(a), we can observe that the number of migrated switches increases while decreasing Th, due to the difference in the load between the set of controllers. However, giving small values

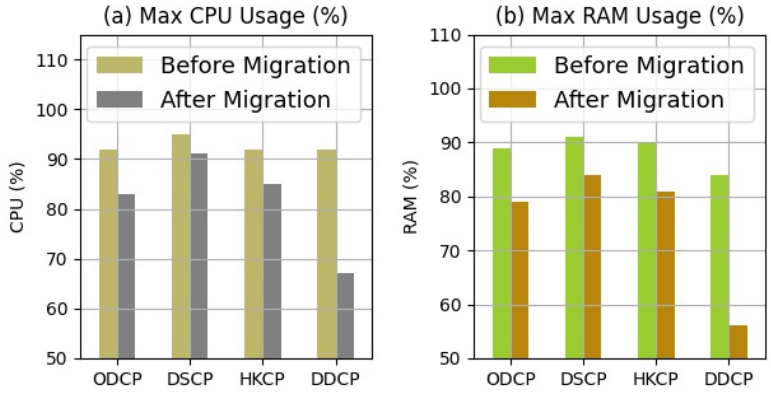

Fig. 6.11 Average resource utilization before and after switch migration

to Th may impact network performances since the migration will happen more frequently in this case, which increases the control delay, as shown in Fig. 6.10(b). On the other hand, when Th is high, the number of switches to migrate is low. However, clusters will be unbalanced in this case since some controllers will be overloaded compared to the others. This results in increasing again but more significantly the control delay, as shown in Fig. 6.10(b). A trade-off between the clusters' load and the number of switches to migrate is thus necessary. According to Fig. 6.10, this trade-off is achieved when the Threshold is equal to 30%. Hence, in our subsequent experiments, we fixed Th to this obtained value.

Fig. 6.11 shows the average resource utilization in terms of CPU and RAM before and after switch migration under all schemes. First, we can observe that, the migration process reduces considerably the average resource utilization for all schemes. The gain is more significant when using our DDCP approach, since it considers both control and data plane performance metrics (i.e., CD, CL, ICD and ICT) in the reward function when deploying a new cluster, as opposed to the other schemes. Indeed, the DDCP approach shows a decrease in CPU usage (respectively, RAM usage) of approximately 24% (respectively, 28%). Compared to ODCP, DSCP, and HKCP, these gains are reduced to 10%, 5%, and 7%, respectively, for the CPU usage. On the other hand, for the RAM usage, these gains are reduced to 10%, 7%, and 9%, respectively. However, we note here that this implies an additional deployment of a cluster (controller) in our DDCP approach since the number of clusters after migration is increased to 6 in our experiment. In contrast, the three other approaches ODCP, DSCP and HKCP keep using the same number of clusters already defined by the network operator.

Fig. 6.12 depicts the number of migrated switches for all schemes. We can see that both ODCP and DDCP reduce the number of migrated switches. On the other hand, this number is higher in the two remaining approaches (i.e. DSCP and HKCP), impacting thus the robustness of the network. In fact, having a high number of switches to migrate increases the signalling overhead, impacting thus the controllers' load and the network stability.

To further show the benefit of our DDCP approach, we plot in Fig. 6.13 the delay-throughput performance metrics (i.e., CD, ICD and ICT) for all schemes. We can see that the DSCP scheme increases the intra-cluster delay and decreases the throughput, since those metrics are not considered when clustering the network. On the other hand, both HKCP and ODCP schemes show better performances, as they take into account additional parameters such as the delay between the controllers. Finally, we can see that our DDCP approach outperforms all other schemes, showing higher throughput and lower delay compared to the others, thanks to the use of the DQN agent with a more complete reward function to solve the controllers' placement problem. However, this comes at the expense of an additional deployment of a cluster/controller in the control plane, as stated previously. It is worth noting that, as several controllers need to be deployed in several locations, the network will be more susceptible to different security challenges and threats.

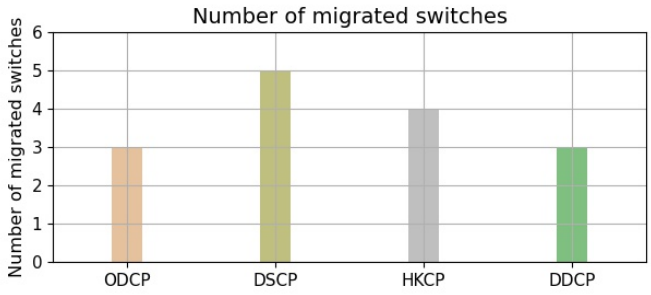

Fig. 6.12 Comparison of number of migrated switches under ODCP, DSCP, HKCP and DDCP schemes

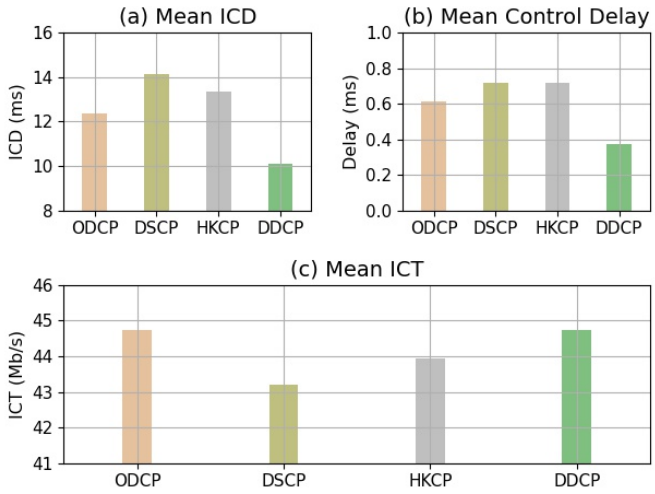

Fig. 6.13 Delay-Throughput metrics evaluation under ODCP, DSCP, HKCP and DDCP schemes

6.5 Conclusion

How many controllers to use in the control plane, where to place them, which switch in the data plane must be controlled by which controller represent challenging questions in SDN. To address these important questions, we used optimization techniques to determine the optimal number of controllers, their optimal placements and the optimal clustering of data plane switches. Because the formulated optimization problem is NP-Hard, a simple yet computationally efficient heuristic algorithm, called DDCP, was proposed and implemented. Our approach solution approach can be used as part of the knowledge plane to optimize control and data plane operations, by deploying a DQN agent that dynamically determines the optimal policy mapping the set of states (clusters of switches) to the set of actions (the set of controllers in specific locations). Experimental results, showed the effectiveness of our approach in identifying the appropriate number of controllers to be deployed and the clustering of data plane switches around these controllers. Moreover, our experiments showed that, the DQN agent outperforms the well known K-means clustering method as well as three main methods proposed in the literature by decreasing the control delay, the control load, and the intra-cluster delay and increasing the intra-cluster throughput. However, this comes at the expense of an additional deployment of a cluster/controller in the control plane.

Chapter 7

Conclusion and Future Work

Contents

7.1 Conclusion

SOFTWARE Defined Networks (SDN), Network Function virtualization (NFV), Network Slicing, Cloud Computing (CC) and Machine Learning (ML) are the key enablers towards the fifth generation (5G) networks. More specifically, combining ML with these technologies to achieve automation and high performances has gaining momentum in the last few years, due to the availability of a large amount of data, the global view of the network as well as its separation from the hardware. In this way, we addressed, in this thesis, QoS-aware routing, Network Slicing and SDN controller placement by taking advantages from ML techniques in a context of a physical and emulated testbeds. After a detailed overview of the main concepts used in this thesis such as 5G requirements and challenges, 5G architecture, SDN and ML, as well an overview of the approaches found in the literature regarding our three axes of research identified in this thesis, we describe our three contributions.

Our first contribution was to present a method for rules placement following the Knowledge Defined Networking (KDN) paradigm, in which the network routing is dynamically optimized by deploying in one side a Deep Q-Network (DQN) agent that dynamically determines the optimal policy mapping the set of states (Traffic Matrices) to the set of actions (changing the vector of link weights) and on the other side, a Traffic Prediction module based on three prediction methods LSTM, ARIMA and LR on the Knowledge Plane (KP), in order to avoid congestion. Thereafter, we mathematically formulated the problem of minimization of the total delay, packet loss and link utilization as a linear program (LP). Then, we proposed a heuristic called DQN and Traffic Prediction based Routing Optimization (DTPRO) that dynamically interacts with the KP, DQN and Traffic Prediction modules. Results showed that our approaches are able to achieve significant gains in terms of routing optimization. Moreover, by combining DQN with Traffic Prediction, we showed that delay, packet loss and link utilization can be decreased, more specifically with the prediction method LSTM, that achieved a high estimation accuracy compared to traditional prediction methods ARIMA and LR.

As a second contribution for this thesis, we have addressed the design and implementation of a novel architecture based on SDN and Machine Learning techniques for enabling creation, modification and continuity of radio and transport network slices, while considering their performances and QoS requirements. To this end, we have firstly incorporated the SDN paradigm to an 5G architecture in both the Radio Access Network (RAN) through FlexRAN and the Packet Network through ONOS. Moreover, we have extended a version of OVS that we patched to handle GTP packets. Thereafter, we mathematically formulated the sharing of RAN and PN resources as a LP, where the optimization problem is to minimize the total network delay, link utilization and packet loss. To solve the formulated problem, we have proposed a Congestion prediction and Load balancing Rule (CLR) placement algorithm. Experimental results showed the feasibility of our proposed solution for handling E2E slicing continuity in real-time. More specifically, the use of ML techniques improves dramatically the performances in terms of latency, packet loss and throughput as compared to classical schemes.

Our third and last contribution focused on the SDN controller placement. We have presented a method that dynamically determines how many controllers to use in the control plane, how to place

them and the corresponding controller for each switch. To this end, we have firstly mathematically modeled the problem. Thereafter, we have proposed and implemented a heuristic called Deep Q-Network based Dynamic Clustering and Placement (DDCP) algorithm following the KDN paradigm, which interacts with a deployed DQN agent that dynamically determines the optimal policy mapping the set of states (clusters of switches) to the set of actions (the set of controllers in specific locations). Results showed the efficiency of using the DQN for controllers and switches clustering and placement as compared to the clustering method K-means, in terms of decreasing the control delay, the control load and the intra-cluster latency.

It is worth noting that in chapter 5, one of our contributions consisted in building a novel testbed for evaluating our proposed approach, due to the lack of implementation from related works. In this way, we built and implemented an E2E 5G platform according to the SDN paradigm, in which we introduced the SD-RAN FlexRAN controller to the access network and the SDN controller ONOS to the transport network of the 5G architecture. Moreover, two real slices have been installed namely eMBB based on mobile connectors as well as an IoT platform to investigate the network slicing concept. Furthermore, the whole testbed was implemented and deployed based on container and orchestration technologies such as Docker and Docker Swarm.

7.2 Future Work

In this thesis, we addressed routing optimization, congestion detection, traffic prediction, network slicing and controller placement in SDN networks. However, there is still work to be done and our contributions can open up further research directions.

In chapter 4, we addressed the routing optimisation and traffic prediction according to the KDN paradigm, where two modules have been deployed in the Knowledge Plane (KP). Moreover, a Network Measurement module is deployed on the control plane to dynamically collect statistics from the data plane. Due to the maturity of SDN technology, there are lack of implementation of SDN in physical switches. As a result, the Network Measurement module feeds the KP module from emulated traffic. It is thus challenging to apply these approaches in real networks. To this end, future directions will consider data collected from a large real networks. Additionally, adding new layers to the SDN architecture in the KDN paradigm might limit the performances in terms of inter-layer delay. As such, considering this factor is also one of our future directions. Finally, in future works the application type will be considered to take decision such as traffic routing and prediction.

In chapter 5, we addressed the creation, modification and continuity of radio and transport network slices in a 5G architecture in both the RAN and the Packet Network. As the traffic is transported in the 5G platform through the GTP protocol, we extended a version of OVS to handle GTP packets. However, this extension did not support the OpenFlow protocol which prevents us from using a large transport network. Extending the OpenFlow protocol in OVS to support GTP traffic will be the target of future works. In addition, the proposed approach supports only two types of slices : eMBB and IoT and these slices are detected in the Packet Network through the slices identifiers not from

inspecting the packet, which can be challenging in the case of several slices. In this way, identifying traffic type based on SDN as well as increasing the number of slices will be considered in future directions. Moreover, the traffic is limited when using the USRP card in the RAN. To this end, the use of collected data from real network can be targeted in future works. Finally, the network slicing is implemented only in the Access and Packet Networks. Implementing Network Slicing in the Core Network is also one of our future perspectives.

We proposed in chapter 6 a method based on DQN to dynamically determine how many controllers to deploy in the control plane, where to place them and their corresponding data plane domains. As a first perspective here, only the K-means method was used to compare with the proposed approach. To this end, more clustering methods will be considered in future works to evaluate the DDCP algorithm. Moreover, combining the proposed solution with the E2E network slicing concept will be targeted in future works.

7.3 Publications

This section lists our published papers within our thesis.

Journals:

1. **E. H. Bouzidi**, A. Outtagarts, R. Langar, and R. Boutaba. Dynamic Clustering of Software Defined Network Switches and Controller placement using Deep Reinforcement Learning. Submitted to Elsevier Computer Networks.

2. **E. H. Bouzidi**, A. Outtagarts, R. Langar, and R. Boutaba. Deep Q-Network and Traffic Prediction based Routing Optimization in Software Defined Networking. submitted to Elsevier Journal of Network and Computer Applications (JNCA), under revision for a second round.

Conferences:

1. **E. H. Bouzidi**, D. Luong, A. Outtagarts, A. Hebbar, and R. Langar. Online-based learning for predictive network latency in software-defined networks. In 2018 IEEE Global Communications Conference (GLOBECOM), pages 1–6, Dec 2018. doi: 10.1109/GLOCOM. 2018. 8648063.

2. **E. H. Bouzidi**, A. Outtagarts, and R. Langar. Deep reinforcement learning application for network latency management in Software Defined Networking. In 2019 IEEE Global Communications Conference (GLOBECOM), pages 1–6, 2019.

3. **E. H. Bouzidi**, A. Outtagarts, A. Hebbar, R. Langar, and R. Boutaba. Online based learning for predictive end-to-end network slicing in 5g networks. In ICC 2020 - 2020 IEEE International Conference on Communications (ICC), pages 1–7, 2020.

Workshops/Demonstrations:

1. **E. H. Bouzidi**, A. Outtagarts, A. Hebbar, and R. Langar. "Slicing Optimisé des Réseaux 5G pour l'Internet des Objets", (FUI SCORPION project, 2020), Nokia Bell Labs France, Paris-Saclay, 7 Route de Villejust, 91620 Nozay, France, 10 Sept. 2019.

Technical Reports:

1. **E. H. Bouzidi**, and A. Outtagarts, "SDN-based dynamic network slicing continuity", (Deliverable 4.3, FUI SCORPION Project), partnering with University Gustave Eiffel, Vertical M2M, Virtual Open Systems, and University La-Rochelle, 23 Aug. 2019.

References

[1] Cloud RAN. Tutorial. URL http://gitlab.eurecom.fr/oai/openairinterface5G/wikis/howto-connect-cots-ue-to-oai-enb-via-ngfi-rru. [Online].

[2] The evolved packet core. Technical report. URL https://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core.

[3] Influxdb. Technical report. URL https://www.influxdata.com/time-series-platform/influxdb/. [Online].

[4] keras. Tutorial. URL https://keras.io/. [Online].

[5] Mongodb. Tutorial. URL https://www.mongodb.com/. [Online].

[6] Cisco NetFlow. http://www.cisco.com/en/US/products/ps6601/. [Online].

[7] Python. Technical report. URL https://www.python.org/. [Online].

[8] Rye. https://osrg.github.io/ryu/resources.html#books. [Online].

[9] Tensorflow. Technical report. URL https://www.tensorflow.org/. [Online].

[10] sflow. https://sflow.org/. [Online].

[11] 3GPP TR 38.801. Study on new radio access technology: Radio access architecture and interfaces. Rel.38.801, Mar. 2016. URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3056. [Online].

[12] 3GPP TS 22.951. Service Aspects and Requirements for Network Sharing. Rel.11, Sep. 2012.

[13] A. Farrel, J.-P. Vasseur, and J. Ash. A path computation element (pce)-based architecture. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf, 2006. [Online].

[14] A. Shang, J. Liao, and L. Du. Pica8 xorplus. http://sourceforge.net/projects/xorplus/, 2014. [Online].

[15] A. Affandi, D. Riyanto, I. Pratomo, and G. Kusrahardjo. Design and implementation fast response system monitoring server using simple network management protocol (snmp). In *2015 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pages 385–390, 2015.

[16] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys & Tutorials*, 20(3):2429–2453, 2018.

[17] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4):2347–2376, 2015.

[18] H. Al-Mohair, J. Mohamad-Saleh, and S. A. Suandi. Hybrid human skin detection using neural network and k-means clustering technique. *Applied Soft Computing*, 33, 05 2015. doi: 10.1016/j.asoc.2015.04.046.

[19] N. Alliance. 5g white paper. *Next generation mobile networks, white paper*, 1, 2015.

[20] M. Azizi, R. Benaini, and M. B. Mamoun. Delay measurement in openflow-enabled mpls-tp network. *Modern Applied Science*, 9(3):90, 2015.

[21] S. Azodolmolky, P. Wieder, and R. Yahyapour. Performance evaluation of a scalable software-defined networking deployment. In *2013 Second European Workshop on Software Defined Networks*, pages 68–74. IEEE, 2013.

[22] A. Azzouni and G. Pujolle. A long short-term memory recurrent neural network framework for network traffic matrix prediction. 05 2017.

[23] B. Pfaff and B. Davie. The open vswitch database management protocol,. http://www.ietf.org/rfc/rfc7047.txt, Dec 2013. [Online].

[24] M. Barabas, G. Boanea, A. B. Rus, V. Dobrota, and J. Domingo-Pascual. Evaluation of network traffic prediction based on neural networks with multi-task learning and multiresolution decomposition. In *2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing*, pages 95–102, Aug 2011. doi: 10.1109/ICCP.2011.6047849.

[25] B. Belter, A. Binczewski, K. Dombek, A. Juszczyk, L. Ogrodowczyk, D. Parniewicz, M. Stroiński, and I. Olszewski. Programmable abstraction of datapath. In *2014 Third European Workshop on Software Defined Networks*, pages 7–12. IEEE, 2014.

[26] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6, 2014.

[27] G. Bianchi, M. Bonola, A. Capone, and C. Cascone. Openstate: programming platform-independent stateful openflow applications inside the switch. *ACM SIGCOMM Computer Communication Review*, 44(2):44–51, 2014.

[28] Big Switch Networks. Project floodlight,. http://www.projectfloodlight.org/, 2013. [Online].

[29] E. Borcoci, T. Ambarus, and M. Vochin. Distributed control plane optimization in sdn-fog vanet. 04 2017.

[30] Bosch XDK Sensor: Cross Domain Development Kit. Bosch xdk sensor: Cross domain development kit,. https://www.bosch-connectivity.com/products/cross-domain/cross-domain-developement-kit/, Jun 2018. [Online].

[31] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16, 2018.

[32] E. H. Bouzidi, A. Outtagarts, R. Langar, and R. Boutaba. Dynamic Clustering of Software Defined Network Switches and Controller placement using Deep Reinforcement Learning. *Submitted to Computer Networks*, .

[33] E. H. Bouzidi, A. Outtagarts, R. Langar, and R. Boutaba. Deep Q-Network and Traffic Prediction based Routing Optimization in Software Defined Networks. *submitted to Elsevier Journal of Network and Computer Applications (JNCA), under revision for a second round*, .

[34] E. H. Bouzidi, D. Luong, A. Outtagarts, A. Hebbar, and R. Langar. Online-based learning for predictive network latency in software-defined networks. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2018. doi: 10.1109/GLOCOM.2018.8648063.

[35] E. H. Bouzidi, A. Outtagarts, and R. Langar. Deep reinforcement learning application for network latency management in software defined networks. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.

[36] E. H. Bouzidi, A. Outtagarts, A. Hebbar, R. Langar, and R. Boutaba. Online based learning for predictive end-to-end network slicing in 5g networks. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–7, 2020.

[37] P. Brockwell and R. A Davis. *An Introduction to Time Series and Forecasting*, volume 39. 01 2002. doi: 10.1007/978-1-4757-2526-1.

[38] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. Merwe. Design and implementation of a routing control platform. 01 2005.

[39] A. Campbell, I. Katzela, K. Miki, and J. Vicente. Open signaling for atm, internet and mobile networks (opensigapos;98). *Computer Communication Review*, 29:97–108, 01 1999. doi: 10.1145/505754.505762.

[40] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. volume 37, pages 1–12, 10 2007. doi: 10.1145/1282380.1282382.

[41] K. Chen and R. Duan. C-RAN: The Road Towards Green RAN, White Paper Version 2.5, China Mobile Research Institute, Oct. 2011, 2011.

[42] W. Chen, C. Chen, X. Jiang, and L. Liu. Multi-controller placement towards sdn based on louvain heuristic algorithm. *IEEE Access*, 6:49486–49497, 2018.

[43] T. Y. Cheng, M. Wang, and X. Jia. Qos-guaranteed controller placement in sdn. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2015.

[44] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–9, 2014.

[45] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski. A knowledge plane for the internet. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–10, 2003.

[46] P. Cortez, M. Rio, M. Rocha, and P. Sousa. Internet traffic forecasting using neural networks. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 2635–2642, 2006.

[47] S. Costanzo, I. Fajjari, N. Aitsaadi, and R. Langar. Dynamic network slicing for 5g iot and embb services: A new design with prototype and implementation results. In *2018 3rd Cloudification of the Internet of Things (CIoT)*, pages 1–7, July 2018. doi: 10.1109/CIOT.2018.8627115.

[48] S. Costanzo, I. Fajjari, N. Aitsaadi, and R. Langar. Demo: Sdn-based network slicing in c-ran. In *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 1–2, Jan 2018. doi: 10.1109/CCNC.2018.8319321.

[49] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, pages 254–265, 2011.

[50] D. Locke. Mq telemetry transport (mqtt) v3. 1 protocol specification,. Http://Www.Ibm.Com/Developerworks/Webservices/Library/Ws-Mqtt/Index.Html, Jun 2010. [Online].

[51] T. Das, V. Sridharan, and M. Gurusamy. A survey on controller placement in sdn. *IEEE Communications Surveys Tutorials*, 22(1):472–503, 2020.

[52] Docker. Docker,. https://www.docker.com, June 2018. [Online].

[53] Docker Swarm. Docker swarm,. https://docs.docker.com/swarm/, Oct 2017. [Online].

[54] E. L. Fernandes and C. E. Rothenberg. Openflow 1.3 software switch, sbrc'2014,. https://github.com/CPqD/ofsoftswitch13, 2014. [Online].

[55] Eclipse Kapua. Eclipse kapua,. https://www.eclipse.org/kapua/, April 2017. [Online].

[56] D. Erickson. The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 13–18, 2013.

[57] ETSI TS 123 401 V13.5.0. Lte; general packet radio service (gprs) enhancements for evolveduniversal terrestrial radio accessnetwork (e-utran) access (3gpp ts 23.401 version 13.5.0 release 13),. http://www.etsi.org/deliver/etsi_ts/123400_123499/123401/13.05.00_60/ts_123401v130500p.pdf, Nov 2016. [Online].

[58] ETSI TS 136 300 V13.2.0. Lte; evolved universal terrestrial radio access (e-utra) and evolved universal terrestrial radio access network (e-utran); overall description; stage 2 (3gpp ts 36.300 version 13.2.0 release 13),. http://www.etsi.org/deliver/etsi_ts/136300_136399/136300/13.02.00_60/ts_136300v130200p.pdf, Nov 2016. [Online].

[59] ETSI's Network Functions Virtualisation (NFV) Industry Specification Group. Network operator perspectives on nfv priorities for 5g. White paper, Feb. 2017.

[60] N. Feamster, J. Rexford, and E. Zegura. The road to sdn: An intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44:87–98, 04 2014. doi: 10.1145/2602204.2602219.

[61] Floodlight. Floodlight project,. http://www.projectfloodlight.org/, Aug 2017. [Online].

[62] N. Foster, M. J. Freedman, R. Harrison, J. Rexford, M. L. Meola, and D. Walker. Frenetic: a high-level language for openflow networks. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, pages 1–6, 2010.

[63] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis. Flexran: A flexible and programmable platform for software-defined radio access networks. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 427–441, 2016.

[64] X. Foukas, M. K. Marina, and K. Kontovasilis. Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture. In *Proceedings of the 23rd annual international conference on mobile computing and networking*, pages 127–140, 2017.

[65] G. Fung. A comprehensive overview of basic clustering algorithms. 2001.

[66] A. A. Gebremariam, M. Usman, P. Du, A. Nakao, and F. Granelli. Towards e2e slicing in 5g: a spectrum slicing testbed and its extension to the packet core. In *2017 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2017.

[67] A. Greenberg, G. Hjálmtýsson, D. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4d approach to network control and management. *ACM SIGCOMM Computer Communication Review*, 35:41–54, 10 2005. doi: 10.1145/1096536.1096541.

[68] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[69] A. Gudipati, L. E. Li, and S. Katti. Radiovisor: A slicing plane for radio access networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 237–238, 2014.

[70] Y. Han, S. Seo, J. Li, J. Hyun, J. Yoo, and J. W. Hong. Software defined networking-based traffic engineering for data center networks. In *The 16th Asia-Pacific Network Operations and Management Symposium*, pages 1–6, Sep. 2014. doi: 10.1109/APNOMS.2014.6996601.

[71] S. Hassas Yeganeh and Y. Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 19–24, 2012.

[72] H. Hejazi, H. Rajab, T. Cinkler, and L. Lengyel. Survey of platforms for massive iot. In *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, pages 1–8. IEEE, 2018.

[73] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker. Practical declarative network management. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 1–10, 2009.

[74] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[75] Y. Hu, T. Luo, N. C. Beaulieu, and C. Deng. The energy-aware controller placement problem in software defined networks. *IEEE Communications Letters*, 21(4):741–744, 2017.

[76] Y.-n. HU, W. Wang, X. Gong, X.-r. QUE, and S. Cheng. On the placement of controllers in software-defined networks. *The Journal of China Universities of Posts and Telecommunications*, 19:92–97, 171, 10 2012. doi: 10.1016/S1005-8885(11)60438-X.

[77] Iperf. Iperf,. http://sourceforge.net/, Marsh 2014. [Online].

[78] M. Karakus and A. Durresi. A survey: Control plane scalability issues and approaches in software-defined networking (sdn). *Computer Networks*, 112:279–293, 2017.

[79] E. Karpilovsky, M. Caesar, J. Rexford, A. Shaikh, and J. van der Merwe. Practical network-wide compression of ip routing tables. *IEEE Transactions on Network and Service Management*, 9(4):446–458, 2012.

[80] K. Katsalis, N. Nikaein, E. Schiller, A. Ksentini, and T. Braun. Network slices toward 5g communications: Slicing the lte network. *IEEE Communications Magazine*, 55(8):146–154, Aug 2017. ISSN 0163-6804. doi: 10.1109/MCOM.2017.1600936.

[81] S. Kaur, J. Singh, and N. Ghumman. Network programmability using pox controller. 08 2014. doi: 10.13140/RG.2.1.1950.6961.

[82] Kavitha S, Varuna S, and Ramya R. A comparative analysis on linear regression and support vector regression. In *2016 Online International Conference on Green Engineering and Technologies (IC-GET)*, pages 1–5, 2016.

[83] S. Khodayari and M. J. Yazdanpanah. Network routing based on reinforcement learning in dynamically changing networks. In *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*, pages 5 pp.–366, Nov 2005. doi: 10.1109/ICTAI.2005.91.

[84] M. G. Kibria, K. Nguyen, G. P. Villardi, K. Ishizu, and F. Kojima. Next generation new radio small cell enhancement: architectural options, functionality and performance aspects. *IEEE Wireless Communications*, 25(4):120–128, 2018.

[85] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*, 2015.

[86] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[87] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[88] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. 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.

[89] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 55–60, 2013.

[90] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1): 14–76, 2015.

[91] A. Ksentini and N. Nikaein. Toward enforcing network slicing on ran: Flexibility and resources abstraction. *IEEE Communications Magazine*, 55(6):102–108, 2017.

[92] A. Ksentini, M. Bagaa, and T. Taleb. On using sdn in 5g: The controller placement problem. In *2016 IEEE GLOBECOM*, pages 1–6, Dec 2016. doi: 10.1109/GLOCOM.2016.7842066.

[93] H. Kuang, Y. Qiu, R. Li, and X. Liu. A hierarchical k-means algorithm for controller placement in sdn-based wan architecture. In *2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, pages 263–267, 2018. doi: 10.1109/ICMTMA.2018.00070.

[94] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann. Heuristic approaches to the controller placement problem in large scale sdn networks. *IEEE Transactions on Network and Service Management*, 12(1):4–17, 2015.

[95] J. Little and S. Graves. *Little's Law*, pages 81–100. 07 2008. doi: 10.1007/978-0-387-73699-0_5.

[96] W. Liu. Intelligent routing based on deep reinforcement learning in software-defined datacenter networks. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2019.

[97] Y. Liu, H. Gu, X. Yu, and J. Zhou. Dynamic sdn controller placement in elastic optical datacenter networks. In *2018 Asia Communications and Photonics Conference (ACP)*, pages 1–3, 2018. doi: 10.1109/ACP.2018.8596219.

[98] A. Lucent. The lte network architecture—a comprehensive tutorial. *Strategic Whitepaper,*, 2009.

[99] M. Smith et al. Opflex control protocol, internet engineering task force,. http://tools.ietf.org/html/draft-smith-opflex-00, Apr 2014. Internet Draft, [Online].

[100] L. Mamushiane, J. Mwangama, and A. A. Lysko. Given a sdn topology, how many controllers are needed and where should they go? In *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–6, 2018.

[101] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *Computer Communication Review*, 38:69–74, 04 2008. doi: 10.1145/1355734.1355746.

[102] J. Merwe and I. Leslie. Switchlets and dynamic virtual atm networks. pages 355–368, 01 1997. doi: 10.1007/978-0-387-35180-3_27.

[103] J. Merwe, S. Rooney, L. Leslie, and S. Crosby. The tempest - a practical framework for network programmability. *Network, IEEE*, 12:20 – 28, 06 1998. doi: 10.1109/65.690958.

[104] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, et al. Knowledge-defined networking. *ACM SIGCOMM Computer Communication Review*, 47(3):2–10, 2017.

[105] Ming-Hung Wang, Shao-You Wu, Li-Hsing Yen, and Chien-Chao Tseng. Pathmon: Path-specific traffic monitoring in openflow-enabled networks. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 775–780, 2016.

[106] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. 12 2013.

[107] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015. doi: 10.1038/nature14236.

[108] H. Mostafaei, M. Menth, and M. S. Obaidat. A learning automaton-based controller placement algorithm for software-defined networks. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2018.

[109] N. Mouawad, R. Naja, and S. Tohme. Optimal and dynamic sdn controller placement. In *2018 International Conference on Computer and Applications (ICCA)*, pages 1–9, 2018. doi: 10.1109/COMAPP.2018.8460361.

[110] R. Mushtaq. Augmented dickey fuller test. *SSRN Electronic Journal*, 08 2011. doi: 10.2139/ssrn.1911068.

[111] L. F. Müller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, and M. P. Barcellos. Survivor: An enhanced controller placement strategy for improving sdn survivability. In *2014 IEEE Global Communications Conference*, pages 1909–1915, 2014.

[112] A. Nakao, P. Du, Y. Kiriha, F. Granelli, A. A. Gebremariam, T. Taleb, and M. Bagaa. End-to-end network slicing for 5g mobile networks. *Journal of Information Processing*, 25:153–163, 2017.

[113] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet. Openairinterface: A flexible platform for 5g research. *ACM SIGCOMM Computer Communication Review*, 44(5):33–38, 2014.

[114] A. Y. Nikravesh, S. A. Ajila, C.-H. Lung, and W. Ding. Mobile network traffic prediction using mlp, mlpwd, and svm. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 402–409. IEEE, 2016.

[115] B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys Tutorials*, 16(3):1617–1634, 2014.

[116] ONF. Software-defined networking: The new norm for networks,. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, April 2012. [Online].

[117] ONF. Open networking foundation,. https://www.opennetworking.org, March 2015. [Online].

[118] ONF. Openflow switch specification,. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, March 2015. Version 1.5.1, [Online].

[119] Open Onos community. Open network operating system (onos),. https://wiki.onosproject.org/display/ONOS/ONOS, Feb 2020. [Online].

[120] Open vSwitch. http://vswitch.org/, 2013. [Online].

[121] Open vSwitch. Open vswitch,. https://docs.openvswitch.org/en/latest/intro/what-is-ovs/, Marsh 2014. [Online].

[122] OpenDaylight. A linux foundation collaborative project,. http://www.opendaylight.org, 2013. [Online].

[123] OpenFlow Community. Switching reference system,. http://www.openflow.org/wp/downloads/, 2009. [Online].

[124] Organization E. Kura framework,. http://www.eclipse.org/kura/, April 2017. [Online].

[125] D. Parniewicz, R. Doriguzzi Corin, L. Ogrodowczyk, M. Rashidi Fard, J. Matias, M. Gerola, V. Fuentes, U. Toseef, A. Zaalouk, B. Belter, et al. Design and implementation of an openflow hardware abstraction layer. In *Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing*, pages 71–76, 2014.

[126] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[127] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, et al. The design and implementation of open vswitch. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 117–130, 2015.

[128] Q. Pham Tran Anh, Y. Hadjadj-Aoul, and A. Outtagarts. *Deep Reinforcement Learning Based QoS-Aware Routing in Knowledge-Defined Networking: Potentiale und Grenzen in der Aus- und Weiterbildung studentischer Tutorinnen und Tutoren*, pages 14–26. 01 2019. ISBN 978-3-658-25802-3. doi: 10.1007/978-3-030-14413-5_2.

[129] C. Qiu, Y. Zhang, Z. Feng, P. Zhang, and S. Cui. Spatio-temporal wireless traffic prediction with recurrent neural network. *IEEE Wireless Communications Letters*, 7(4):554–557, 2018.

[130] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts. A deep reinforcement learning approach for vnf forwarding graph embedding. *IEEE Transactions on Network and Service Management*, 16(4):1318–1331, 2019.

[131] S. Rashid and S. Razak. Big data challenges in 5g networks. pages 152–157, 07 2019. doi: 10.1109/ICUFN.2019.8806076.

[132] M. Rayani, D. Naboulsi, R. Glitho, and H. Elbiaze. Slicing virtualized epc-based 5g core network for content delivery. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 00726–00729. IEEE, 2018.

[133] M. R. Raza, M. Fiorani, A. Rostami, P. Öhlen, L. Wosinska, and P. Monti. Dynamic slicing approach for multi-tenant 5g transport networks. *IEEE J. Opt. Commun. Netw.*, 10(1):A77–A90, Jan 2018. ISSN 1943-0620. doi: 10.1364/JOCN.10.000A77.

[134] M. Reza Parsaei, R. Mohammadi, and R. Javidan. A new adaptive traffic engineering method for telesurgery using aco algorithm over software defined networks. *European Research in Telemedicine / La Recherche EuropÃ©enne en TÃ©lÃ©mÃ©decine*, 6, 11 2017. doi: 10.1016/j.eurtel.2017.10.003.

[135] L. Richardson and S. Ruby. *RESTful Web Services*. Sebastopol, CA, USA:O'Reilly Media, 2008.

[136] N. U. Roiha, Y. K. Suprapto, and A. D. Wibawa. The optimization of the weblog central cluster using the genetic k-means algorithm. In *2016 International Seminar on Application for Technology of Information and Communication (ISemantic)*, pages 278–284, 2016.

[137] M. R. Sama, S. Ben Hadj Said, K. Guillouard, and L. Suciu. Enabling network programmability in lte/epc architecture using openflow. pages 389–396, 05 2014. ISBN 978-3-901882-63-0. doi: 10.1109/WIOPT.2014.6850324.

[138] M. Series. Minimum requirements related to technical performance for imt-2020 radio interface (s). *Report*, pages 2410–0, 2017.

[139] P. Sharma. Evolution of mobile wireless communication networks-1g to 5g as well as future prospective of next generation communication network. *International Journal of Computer Science and Mobile Computing*, 2(8):47–53, 2013.

[140] D. Sheinbein and R. Weber. Stored program controlled network : 800 service using spc network capability. *Bell System Technical Journal, The*, 61:1737–1744, 09 1982. doi: 10.1002/j.1538-7305.1982.tb04370.x.

[141] S. Shenker. Fundamental design issues for the future internet. *IEEE Journal on selected areas in communications*, 13(7):1176–1188, 1995.

[142] H. Song. Protocol-oblivious forwarding: unleash the power of sdn through a future-proof forwarding plane. 08 2013. doi: 10.1145/2491185.2491190.

[143] Y. Su, R. Fan, X. Fu, and Z. Jin. Dqelr: An adaptive deep q-network-based energy- and latency-aware routing protocol design for underwater acoustic sensor networks. *IEEE Access*, 7:9091–9104, 2019.

[144] M. Suñé, V. Alvarez, T. Jungel, U. Toseef, and K. Pentikousis. An openflow implementation for network processors. In *2014 Third European Workshop on Software Defined Networks*, pages 123–124. IEEE, 2014.

[145] M. M. Tajiki, B. Akbari, and N. Mokari. Qrtp:qos-aware resource reallocation based on traffic prediction in software defined cloud networks. In *2016 8th International Symposium on Telecommunications (IST)*, pages 527–532, Sep. 2016. doi: 10.1109/ISTEL.2016.7881877.

[146] M. Tanha, D. Sajjadi, and J. Pan. Enduring node failures through resilient controller placement for software defined networks. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, 2016.

[147] P. Tao, C. Ying, Z. Sun, S. Tan, P. Wang, and Z. Sun. The controller placement of software-defined networks based on minimum delay and load balancing. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing*, pages 310–313, 2018.

[148] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. In *Proceedings DARPA Active Networks Conference and Exposition*, pages 2–15, 2002.

[149] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, 1997.

[150] A. Tootoonchian and Y. 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.

[151] Universal Software Radio Peripheral USRP. Universal software radio peripheral usrp,. https://www.ettus.com/, Oct 2017. [Online].

[152] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers. Opennetmon: Network monitoring in openflow software-defined networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–8, 2014.

[153] A. Voellmy and P. Hudak. Nettle: A language for configuring routing networks. In *IFIP Working Conference on Domain-Specific Languages*, pages 211–235. Springer, 2009.

[154] G. Wang, Y. Zhao, J. Huang, Q. Duan, and J. Li. A k-means-based network partition algorithm for controller placement in software defined network. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, 2016.

[155] Wikipedia. Ordinary least squares,. https://en.wikipedia.org/wiki/Ordinary_least_squares, Aug 2020. [Online].

[156] XDK Workbench. Xdk workbench,. https://xdk.bosch-connectivity.com/, April 2017. [Online].

[157] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie. A survey on software-defined networking. *IEEE Communications Surveys Tutorials*, 17(1):27–51, 2015.

[158] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu. A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges. *IEEE Communications Surveys Tutorials*, 21(1):393–430, 2019.

[159] Y. Xu, F. Yin, W. Xu, J. Lin, and S. Cui. Wireless traffic prediction with scalable gaussian process: Framework, algorithms, and verification. *IEEE Journal on Selected Areas in Communications*, 37(6):1291–1306, 2019.

[160] Y. Yiakoumis, J. Schulz-Zander, and J. Zhu. Pantou: Openflow 1.0 for openwrt,. http://www.openflow.org/wk/index.php/Open_Flow1.0_forOpenWRT, 2011. [Online].

[161] Y. Yamada, R. Shinkuma, T. Sato, and E. Oki. Feature-selection based data prioritization in mobile traffic prediction using machine learning. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[162] L. Yang, T. Anderson, R. Dantu, and R. Gopal. Forwarding and control element separation (forces) framework. 05 2004.

[163] G. Yao, J. Bi, Y. Li, and L. Guo. On the capacitated controller placement problem in software defined networks. *IEEE Communications Letters*, 18(8):1339–1342, 2014.

[164] C. Yu, C. Lumezanu, A. Sharma, Q. Xu, G. Jiang, and H. V. Madhyastha. Software-defined latency monitoring in data center networks. In *International Conference on Passive and Active Network Measurement*, pages 360–372. Springer, 2015.

[165] C. Yu, J. Lan, Z. Guo, and Y. Hu. Drom: Optimizing the routing in software-defined networks with deep reinforcement learning. *IEEE Access*, 6:64533–64539, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2877686.

[166] H. Zhang, N. Liu, X. Chu, K. Long, A. Aghvami, and V. C. M. Leung. Network slicing based 5g and future mobile networks: Mobility, resource management, and challenges. *IEEE Communications Magazine*, 55(8):138–145, Aug 2017. ISSN 0163-6804. doi: 10.1109/MCOM.2017.1600940.

[167] Y. Zhao, Y. Li, X. Zhang, G. Geng, W. Zhang, and Y. Sun. A survey of networking applications applying the software defined networking concept based on machine learning. *IEEE Access*, 7:95397–95417, 2019.