Network reconfiguration and management in 6G telecommunication networks

by

Theodoros Tsourdinis

Supervisors:

Serge Fdida, Professor, Sorbonne Université
Thanasis Korakis, Professor, University of Thessaly

Committee:

Serge Fdida, Professor, Sorbonne Université (Supervisor)
Thanasis Korakis, Professor, University Of Thessaly (Supervisor)
Walid Dabbous, Professor, INRIA (Reviewer)
Thi-Mai-Trang Nguyen, Professor, Universite Sorbonne Paris Nord (Reviewer)
Jim Kurose, Professor, University of Massachusetts (Examiner)
Anne Fladenmuller, Professor, Sorbonne Université (Examiner)
Paris Flegkas, Assistant Professor, University of Thessaly (Examiner)

October 2025

UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Network reconfiguration and management in 6G telecommunication networks

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Theodoros Tsourdinis

October 2025

UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Reconfiguration et gestion des réseaux de télécommunication 6G

Thèse présentée pour l'obtention du diplôme de Doctorat

Theodoros Tsourdinis

October 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Δυναμική προσαρμογή και διαχείριση τηλεπικοινωνιακών δικτύων 6ης γενιάς

Διατριβή η οποία υποβλήθηκε για τη μερική εκπλήρωση
των υποχρεώσεων απόκτησης του Διδακτορικού Διπλώματος

Θεόδωρος Τσουρδίνης

Οκτώβριος 2025

Network reconfiguration and management in 6G telecommunication networks

PhD Dissertation

Theodoros Tsourdinis

Advisory committee

Serge Fdida, Professor, Sorbonne Université (Supervisor)

Thanasis Korakis, Professor, University Of Thessaly (Supervisor)

Examination committee

Walid Dabbous, Professor, INRIA (Reviewer)

Thi-Mai-Trang Nguyen, Professor, Universite Sorbonne Paris Nord (Reviewer)

Paris Flegkas, Assistant Professor, University of Thessaly (Examiner)

Jim Kurose, Professor, University of Massachusetts (Examiner)

Anne Fladenmuller, Professor, Sorbonne Université (Examiner)

October 2025

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this PH.D. dissertation, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

The declarant

Theodoros Tsourdinis

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisors, Professor Thanasis Korakis from the University of Thessaly and Professor Serge Fdida from Sorbonne University.

Professor Korakis gave me the invaluable opportunity to enter into the world of research and pursue my doctoral as part of his distinguished team at NITLab. Throughout this journey his support in broadening my research perspectives have been invaluable and I am truly greatful for his guidance and mentorship.

Professor Fdida provided exceptional hospitality and outstanding guidance throughout my research period at Sorbonne and the LIP6 laboratory. His visionary thinking, openness to new ideas, and constant encouragement played a pivotal role in my evolution as a researcherer. He offered me not only the freedom to explore various research topics but also continuous access to a rich ecosystem of academic collaboration, training programs, and international exposure. Working under his supervision and mentorship significantly expanded my perspective on how to approach advanced research projects in a collaborative and impactful manner.

I would like to sincerely thank Dr. Nikos Makris, who supported my entrance into the research world. As a highly skilled postdoctoral researcher, he played a key role in shaping both the direction and quality of my doctoral research. His expertise and ideas significantly influenced my work. In addition to his scientific contributions, he offered consistent support and mentorship throughout my PhD journey, for which I am truly grateful.

I would also like to extend my sincere appreciation to my collaborators and colleagues at NITLab: Ilias Chatzistefanidis, Sokratis Christakis, Vasilis Zalokostas, Dimitris Kefalas, Apostolis Prassas, Katerina Kyriakou, Ipporatis Koukoulis and Efraim Pavlidis as well as the postdoctoral researchers including Virgilios Passas, Apostolis Apostolaras, Ilias Syrigos, Kostas Chounos and Kostas Choumas. Additionally, my heartfelt thanks to my colleagues

at LIP6: Hassane Rahich, Frederic Vaissade, Albert SU, Taha Mohsen, Bilel Zaghoudi, and Anastasios Giovanidis.

A special acknowledgment to Emilie Mespoulhes for her invaluable administrative assistance and essential support during my relocation to Paris. Without her timely and diligent help, my transition to Paris and progression through the doctoral program would have been significantly more challenging.

I also express my gratitude to the administrative team at NITLab, specifically Stavroula Maglavera, Katerina Arvaniti, Sissy Magouritsa, Olina Stergiopoulou, and Stergios Avlonitis, for their support and assistance with administrative matters.

My heartfelt gratitude goes to my family: my parents, Georgios Tsourdinis and Sofia Mavromatopoulou, and my sister, Anna Tsourdini. Their endless sacrifices, financial support, relocations, and emotional resilience provided the foundation necessary for me to achieve this significant milestone in my educational journey.

I would also like to deeply thank my close friends who stood by me in both joyful and challenging times: Aggelos Argyriou, Dimitris Monias, Ilias Chatzistefanidis, Kyriakos Kallergis, Thanasis Panagos, Serafeim Gikas, Sokratis Christakis and all my other dear friends.

Above all, I am deeply thankful to Jesus Christ.

PhD Dissertation

Network reconfiguration and management in 6G telecommunication networks

Theodoros Tsourdinis

Abstract

Fifth-generation (5G) telecommunication networks promise unprecedented improvements in connectivity, offering ultra-high data rates, ultra-reliable low-latency communication (URLLC), and massive connectivity of diverse devices. These capabilities are essential for enabling transformative applications such as autonomous vehicles, remote surgery, industrial automation, and the Internet of Things (IoT). However, despite the theoretical advances, practical deployments frequently fail to meet anticipated performance benchmarks. This performance gap primarily arises from simplistic assumptions regarding user mobility patterns, static resource allocation strategies, and limited adaptability to changing network conditions. As the industry transitions toward sixth-generation (6G) networks, addressing these challenges through dynamic reconfiguration and advanced management mechanisms becomes critically important.

This thesis investigates fundamental questions related to dynamic resource allocation, seamless service continuity during user mobility, and robust security in highly automated edge network environments. Specifically, the research addresses the following key problems: how can Artificial Intelligence and Machine Learning (AI/ML) techniques be leveraged to dynamically enhance and optimize resource allocation within the Radio Access Network (RAN)? How can low-latency access be continuously maintained for mobile users within Multi-Access Edge Computing (MEC) frameworks? Lastly, how can the security and resilience of edge network infrastructures be guaranteed against evolving threats such as Denial of Service (DoS) attacks?

To tackle dynamic resource allocation challenges, we designed and incorporated an MLOps platform within the 5G architecture, which collects user and radio data through a custom Network Data Analytics Function (NWDAF) module. By employing deep learning techniques,

we predict user demands, application interactions, and radio conditions, enabling proactive reallocation of Physical Resource Blocks (PRB). This approach demonstrates a successful transition from traditional network-aware applications—where services must adapt to network conditions—to a novel service-aware paradigm, wherein the network autonomously aligns itself with real-time application needs, significantly reducing resource over- and under-provisioning.

To ensure seamless service continuity in MEC environments, we introduced a heterogeneous connectivity framework integrating both 3GPP and non-3GPP technologies. We developed a service migration controller that dynamically selects optimal Radio Access Technology (RAT) paths based on real-time radio conditions, that also migrates edge services closer to user locations. Additionally, we proposed and implemented a Deep Reinforcement Learning (DRL)-based migration approach, utilizing multi-cell Round-Trip Time (RTT) measurements to proactively reposition services and maintain continuous, low-latency access during user mobility. Extensive experimental evaluations validated the effectiveness of this approach, demonstrating uninterrupted high-quality user experiences.

Addressing network security, we integrated a robust anomaly detection mechanism within an Open RAN architecture. This system identifies and mitigates real-time security threats such as DoS attacks, dynamically adjusting resource allocations, and manages the users to maintain network integrity and service quality. The synergy of dynamic resource allocation and security enhancements significantly improves the resilience and reliability of next-generation network infrastructures.

Comprehensive experimental evaluations conducted in realistic testbed environments highlight substantial improvements across multiple performance metrics, including reduced latency, increased throughput, optimized resource utilization, and enhanced energy efficiency. These results demonstrate the practical viability and efficacy of the proposed methodologies, providing robust foundations for addressing similar challenges anticipated in emerging 6G network ecosystems.

Keywords

Beyond 5G; service-aware; network slicing; Multi-access Edge Computing; Beyond 5G; Cloud-Native network; AI/ML; OpenAirInterface; Kubernetes;

Διδακτορική Διατριβή

Δυναμική προσαρμογή και διαχείριση τηλεπικοινωνιακών δικτύων 6ης γενιάς

Θεόδωρος Τσουρδίνης

Περίληψη

Τα τηλεπικοινωνιακά δίκτυα πέμπτης γενιάς (5G) υπόσχονται πρωτοφανείς βελτιώσεις στη συνδεσιμότητα, προσφέροντας υψηλούς ρυθμούς δεδομένων, εξαιρετικά αξιόπιστες επικοινωνίες χαμηλής καθυστέρησης (URLLC) και μαζική διασύνδεση ετερογενών συσκευών. Οι δυνατότητες αυτές αποτελούν κρίσιμο καταλύτη για την υποστήριξη εφαρμογών όπως τα αυτόνομα οχήματα, η απομακρυσμένη χειρουργική, ο βιομηχανικός αυτοματισμός και το Διαδίκτυο των Πραγμάτων (IoT). Ωστόσο, οι πραγματικές εγκαταστάσεις συχνά δεν ανταποκρίνονται στις αναμενόμενες επιδόσεις, εξαιτίας απλοϊκών υποθέσεων σχετικά με την κινητικότητα των χρηστών, στατικής κατανομής πόρων και περιορισμένης προσαρμογής σε δυναμικά περιβάλλοντα. Αυτές οι αδυναμίες καθιστούν αναγκαία την ανάπτυξη προηγμένων μηχανισμών δυναμικής αναδιαμόρφωσης και ευφυούς διαχείρισης, καθώς η βιομηχανία προετοιμάζεται για τα δίκτυα έκτης γενιάς (6G).

Η παρούσα διατριβή διερευνά θεμελιώδη ερωτήματα που αφορούν τη δυναμική κατανομή πόρων, τη συνεχή παροχή υπηρεσιών σε κινητούς χρήστες και την ενίσχυση της ασφάλειας σε αυτοματοποιημένα περιβάλλοντα δικτύων. Συγκεκριμένα, η έρευνα απαντά στα εξής κρίσιμα ερωτήματα: Πώς μπορούν οι τεχνικές Τεχνητής Νοημοσύνης και Μηχανικής Μάθησης (AI/ML) να αξιοποιηθούν για τη δυναμική βελτιστοποίηση της κατανομής πόρων στο Δίκτυο Πρόσβασης Ασύρματου Μέσου (ΔΠΑΜ) (RAN); Πώς μπορεί να διατηρείται συνεχής πρόσβαση χαμηλής καθυστέρησης για κινητούς χρήστες σε Περιβάλλοντα Υπολογιστικής Ισχύος στην Άκρη του Δικτύου (ΠΥΙΑΔ) (MEC); Τέλος, πώς μπορεί να διασφαλιστεί η ασφάλεια και ανθεκτικότητα των τηλεπικοινωνιακών δικτύων έναντι απειλών όπως οι επιθέσεις άρνησης υπηρεσίας (ΕΑΥ) (DoS);

Για την βέλτιστη κατανομή πόρων στο ΔΠΑΜ, αναπτύξαμε μια πλατφόρμα Λειτουργικής Διαχείρισης Μηχανικής Μάθησης (MLOps) ενσωματωμένη στην αρχιτεκτονική 5G,

η οποία συλλέγει δεδομένα χρηστών και ΔΠΑΜ μέσω μιας προσαρμοσμένης λειτουργίας Αναλυτικής Επεξεργασίας Δεδομένων Δικτύου (NWDAF). Με τη χρήση μοντέλων βαθιάς μάθησης, προβλέπουμε τη ζήτηση των χρηστών, τις συνθήκες του ΔΠΑΜ και τις απαιτήσεις των εφαρμογών, επιτρέποντας την προληπτική και δυναμική κατανομή των πόρων του RAN. Αυτή η προσέγγιση σηματοδοτεί τη μετάβαση από ένα παραδοσιακό μοντέλο προσαρμογής των εφαρμογών στο δίκτυο (network-aware) σε ένα καινοτόμο μοντέλο προσανατολισμένο στις υπηρεσίες (service-aware), στο οποίο το ίδιο το δίκτυο προσαρμόζεται στις ανάγκες των εφαρμογών σε πραγματικό χρόνο, μειώνοντας δραστικά την υπερ- ή υπο-κατανομή πόρων.

Για την εξασφάλιση συνεχούς πρόσβασης χαμηλής καθυστέρησης σε ΠΥΙΑΔ, αναπτύξαμε μια πλατφόρμα ετερογενούς συνδεσιμότητας που ενσωματώνει τεχνολογίες 3GPP και μη-3GPP. Δημιουργήσαμε έναν ελεγκτή δυναμικής μετεγκατάστασης υπηρεσιών (Service Live Migration), που επιλέγει τη Βέλτιστη Τεχνολογία Πρόσβασης στο Ασύρματο Μέσο (RAT) με βάση τις πραγματικές συνθήκες του δικτύου και αξιοποιεί έναν αλγόριθμο Βαθιάς Ενισχυτικής Μάθησης (DRL) για την προληπτική μετεγκατάσταση των υπηρεσιών MEC πιο κοντά στους χρήστες. Οι πειραματικές αξιολογήσεις επιβεβαίωσαν τη σημαντική βελτίωση της καθυστέρησης πρόσβασης και της συνέχειας των υπηρεσιών κατά την κινητικότητα των χρηστών.

Τέλος, ενσωματώσαμε έναν ισχυρό μηχανισμό ανίχνευσης ανωμαλιών στην αρχιτεκτονική ανοιχτού ΔΠΑΜ (O-RAN), ο οποίος αναγνωρίζει και αντιμετωπίζει απειλές όπως η ΕΑΥ σε πραγματικό χρόνο, διασφαλίζοντας την ακεραιότητα και τη συνεχή ποιότητα των υπηρεσιών.

Οι πειραματικές αξιολογήσεις καταδεικνύουν σημαντικές βελτιώσεις σε καθυστέρηση, διαμεταγωγή, βέλτιστη αξιοποίηση πόρων και ενεργειακή αποδοτικότητα, επιβεβαιώνοντας τη βιωσιμότητα των προτεινόμενων λύσεων, και παρέχουν στέρεες βάσεις για την αντιμετώπιση αντίστοιχων προκλήσεων στα επερχόμενα δίκτυα 6G.

Keywords

Beyond 5G; service-aware; network slicing; Multi-access Edge Computing; Beyond 5G; Cloud-Native network; AI/ML; OpenAirInterface; Kubernetes;

Thèse de Doctorat

Reconfiguration et gestion des réseaux de télécommunication 6G

Theodoros Tsourdinis

Résumé

Les réseaux de télécommunications de cinquième génération (5G) promettent des avancées sans précédent en matière de connectivité, offrant des débits de données extrêmement élevés, des communications ultra-fiables à très faible latence (URLLC) ainsi qu'une connectivité massive pour une diversité d'appareils. Ces caractéristiques sont essentielles pour soutenir des applications critiques telles que les véhicules autonomes, la chirurgie à distance, l'automatisation industrielle et l'Internet des objets (IoT). Cependant, malgré leurs performances théoriques remarquables, les déploiements pratiques peinent souvent à atteindre les indicateurs de performance attendus, principalement en raison d'hypothèses simplistes sur les schémas de mobilité des utilisateurs, d'une allocation statique des ressources réseau, et d'une capacité d'adaptation limitée face aux conditions changeantes du réseau. À mesure que l'industrie évolue vers les réseaux de sixième génération (6G), il devient crucial de résoudre ces défis par des mécanismes avancés de reconfiguration dynamique et de gestion.

Cette thèse examine des questions fondamentales liées à l'allocation dynamique des ressources, à la continuité transparente des services lors de la mobilité des utilisateurs, ainsi qu'à la sécurité robuste des environnements réseau hautement automatisés. Plus précisément, elle répond aux problématiques suivantes : comment les techniques d'intelligence artificielle et d'apprentissage automatique (AI/ML) peuvent-elles être exploitées pour optimiser dynamiquement l'allocation des ressources au sein du réseau d'accès radio (RAN) ? Comment peut-on maintenir en permanence un accès à faible latence pour les utilisateurs mobiles dans des environnements Multi-Access Edge Computing (MEC) ? Enfin, comment assurer la sécurité et la résilience des infrastructures réseau pilotées par l'IA face à des menaces évolutives telles que les attaques par déni de service (DoS) ?

Pour résoudre les défis liés à l'allocation dynamique des ressources, nous avons conçu et intégré une plateforme MLOps au sein de l'architecture 5G. Celle-ci collecte les données utilisateurs et radio via une fonction analytique réseau personnalisée (NWDAF). En utilisant

des techniques d'apprentissage profond (deep learning), nous prédisons les besoins des utilisateurs ainsi que les conditions radio, permettant une réallocation proactive des ressources basée sur ces prédictions. Cette approche permet une transition réussie du modèle traditionnel où les applications doivent s'adapter aux conditions du réseau (network-aware), vers un nouveau paradigme orienté service (service-aware), dans lequel le réseau s'adapte automatiquement aux besoins applicatifs en temps réel, réduisant ainsi significativement la surallocation et la sous-allocation des ressources.

Pour garantir la continuité des services dans les environnements MEC, nous avons développé un cadre de connectivité hétérogène intégrant à la fois les technologies 3GPP et non-3GPP. Nous avons également conçu un contrôleur de migration de services MEC capable de sélectionner dynamiquement le meilleur chemin de technologie d'accès radio (RAT), en utilisant un modèle basé sur l'apprentissage par renforcement profond (DRL). Ce modèle exploite des mesures Round-Trip Time (RTT) provenant de multiples cellules pour relocaliser proactivement les services, garantissant ainsi un accès à faible latence constant lors des déplacements des utilisateurs. Des évaluations expérimentales approfondies ont confirmé l'efficacité de cette approche, assurant une expérience utilisateur continue et de haute qualité.

Concernant la sécurité réseau, nous avons intégré un mécanisme robuste de détection d'anomalies au sein d'une architecture Open RAN. Ce mécanisme identifie et atténue en temps réel les menaces telles que les attaques par déni de service (DoS), en ajustant dynamiquement les ressources réseau pour préserver l'intégrité et la qualité du service. L'intégration de cette sécurité à l'allocation dynamique des ressources améliore significativement la résilience et la fiabilité des infrastructures réseau pilotées par l'IA.

Des évaluations expérimentales complètes menées dans des environnements réalistes démontrent des améliorations substantielles dans plusieurs indicateurs de performance clés, notamment une réduction de la latence, une augmentation du débit, une optimisation de l'utilisation des ressources et une efficacité énergétique accrue. Ces résultats valident la faisabilité pratique et l'efficacité des méthodologies proposées, établissant ainsi des bases solides pour relever des défis similaires et plus complexes dans les futurs réseaux 6G.

En résumé, cette thèse contribue significativement à la réalisation d'infrastructures de télécommunications flexibles, sécurisées et intelligentes, réduisant efficacement l'écart entre les capacités théoriques et les performances opérationnelles réelles des réseaux.

Keywords

Beyond 5G; service-aware; network slicing; Multi-access Edge Computing; Beyond 5G; Cloud-Native network; AI/ML; OpenAirInterface; Kubernetes;

List of Publications

Journals

[J1] Theodoros Tsourdinis, Ilias Chatzistefanidis, Nikos Makris, Thanasis Korakis, Navid Nikaein, Serge Fdida. **Service-aware real-time slicing for virtualized beyond 5G networks**. Computer Networks: The International Journal of Computer and Telecommunications Networking, Volume 247, Issue C (COMNETS), 2024, ACM [1].

Conferences

[C1] Theodoros Tsourdinis, Nikos Makris, & Thanasis Korakis. **Experimental evaluation of a Follow-me MEC Cloud-Native 5G network**. IEEE 4th 5G World Forum (5GWF), 2021, IEEE [2].

[C2] Theodoros Tsourdinis, Ilias Chatzistefanidis, Nikos Makris, & Thanasis Korakis. **AI-driven Service-aware Real-time Slicing for beyond 5G Networks**. IEEE International Conference on Computer Communications (Infocom), 2022, IEEE - **Reproducibility Award** [3].

[C3] Theodoros Tsourdinis, Nikos Makris, Serge Fdida & Thanasis Korakis. **DRL-based Service Migration for MEC Cloud-Native 5G and beyond Networks**. 10th IEEE International Conference on Network Softwarization (NetSoft), 2023, IEEE [4].

[C4] Theodoros Tsourdinis, Nikos Makris, Thanasis Korakis, Serge Fdida. **Demystifying URLLC in Real-World 5G Networks: An End-to-End Experimental Evaluation.** IEEE Global Communications Conferences (Globecom), 2024, IEEE [5].

[C5] Theodoros Tsourdinis, Nikos Makris, Thanasis Korakis, Serge Fdida. **AI-Driven Network Intrusion Detection and Resource Allocation in Real-World O-RAN 5G Net-**

works. The 30th Annual International Conference on Mobile Computing and Networking (Mobicom), 2024, ACM. [6]

[C6] Theodoros Tsourdinis, Nikos Makris, Thanasis Korakis, Serge Fdida. **Real-World Reinforcement Learning for Energy-Efficient DL Power Management in Beyond 5G RAN**. IEEE Infocom (Under Review), 2026, IEEE.

In addition, our research efforts within the same period led to the following publications that are not directly related to this thesis:

Conferences

C1 Nikos Makris, Virgilios Passas, Apostolos Apostolaras, Theodoros Tsourdinis, Ilias Chatzistefanidis & Thanasis Korakis. **On enabling remote hands-on Computer Networking Education: the NITOS testbed approach**. 2023 IEEE Integrated STEM Education Conference, (ISTEM), 2023, IEEE [7].

C2 Socratis Christakis, Theodoros Tsourdinis, Nikos Makris, Thanasis Korakis, Serge Fdida. **Evaluation of User Plane Function Implementations in Real-World 5G Networks**. IEEE International Conference on Computer Communications (Infocom), 2024, IEEE [8].

Table of contents

List of figures

List of tables

Abbreviations

2-6G	2nd–6th generation mobile networks
A1	Interface from non-RT RIC to near-RT RIC
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
AR	Augmented Reality
ARQ	Automatic Repeat reQuest
ATD	Anomaly Traffic Detector
AUC	Area Under Curve
AWGN	Additive White Gaussian Noise
BBU	Baseband Unit
Bi-LSTM	Bidirectional Long Short-Term Memory
BLER	Block Error Rate

BPF	Bandwidth Part
C-RAN	Cloud Radio Access Network
CQI	Channel Quality Indicator
CRAF	Core RAN Analytics Function
CRIU	heckpoint/Restore In Userspace
CP	Control Plane
CPU	Central Processing Unit
CU	Centralized Unit
CU-CP	Centralized Unit – Control Plane
CU-UP	Centralized Unit – User Plane
CUPS	Control and User Plane Separation
CNN	Convolutional Neural Network
CNN-LSTM	Convolutional Neural Network - Long Short-Term Memory
DL	Downlink
DNN	Deep Neural Network
DoS	Denial of Service

DPDK	Data Plane Development Kit
DRL	Deep Reinforcement Learning
DU	Distributed Unit
E2	Interface between near-RT RIC and E2 nodes
E2-Agent	O-RAN E2 Node Agent
eMBB	Enhanced Mobile Broadband
eNB	4G Radio Node B
F1, E1, E2	O-RAN / 3GPP functional split interfaces
FAPI	Fronthaul Application Programming Interface
FDD	Frequency-Division Duplexing
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FNN	Feed-Forward Neural Network
FlexRIC	Flexible RAN Intelligent Controller
FP	False Positive
FR1, FR2	Frequency Range 1 and 2

FTP	File Transfer Protocol
gNB	5G New Radio Node B
GHz	Gigahertz
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
GTP	GPRS Tunneling Protocol
HARQ	Hybrid Automatic Repeat Request
HTTP	Hypertext Transfer Protocol
HSS	Home Subscriber Server
IDS	Intrusion Detection System
IMT	International Mobile Telecommunications
ISAC	Integrated Sensing and Communications
ITU	International Telecommunication Union
JCAS	Joint Communications and Sensing
K8s	Kubernetes
KDDCUP'99	Knowledge Discovery and Data Mining Cup 1999 Dataset

KNN	K-Nearest Neighbors
LMF	Location Management Function
LOF	Local Outlier Factor
LSTM	Long Short-Term Memory
MAC	Medium Access Control
MAE	Mean Absolute Error
Mbps	Megabits per second
MDP	Markov Decision Processes
MEC	Multi-Access Edge Computing
MinMaxScaler	Minimum-Maximum Normalization Scaler
MIMO	Multiple Input Multiple Output
ML	Machine Learning
MLOps	Machine Learning Operations
MME	Mobility Management Entity
mMTC	massive Machine-Type Communications
MySQL	Structured Query Language Database

NAS	Non-Access Stratum
NF, NFV	Network Function; Network Functions Virtualization
NGFI	Next Generation Fronthaul Interface
NGINX	Open-source Web Server
NITOS	Network Implementation Testbed
NR	New Radio (5G)
NSSAI	Network Slice Selection Assistance Information
NSSF	Network Slice Selection Function
NWDAF	Network Data Analytics Function
O-RAN	Open Radio Access Network
OAI	OpenAirInterface
OFDM	Orthogonal Frequency-Division Multiplexing
PDCP	Packet Data Convergence Protocol
PDU	Protocol Data Unit
PHY	Physical Layer
PRB	Physical Resource Block

PS	Parameter Server
PUCCH	Physical Uplink Control Channel
PUSCH	Physical Uplink Shared Channel
PyShark	Python wrapper for TShark network protocol analyzer
QoE	Quality of Experience
QoS	Quality of Service
R2	Coefficient of Determination
RAN	Radio Access Network
RAT	Radio Access Technology
RB	Resource Block
ReLU	Rectified Linear Unit
RF	Random Forest
RFC	Random Forest Classifier
RIC	RAN Intelligent Controller
RL	Reinforcement Learning
RLC	Radio Link Control

ROC	Receiver Operating Characteristic
RRC	Radio Resource Control
RTT	Round-Trip Time
S-NSSAI	Single Network Slice Selection Assistance Information
SBA	Service-Based Architecture
Scapy	Python-based Packet Manipulation Tool
SD	Slice Differentiator
SGi	Interface between GGSN/UPF and the external PDN
SGW	Serving Gateway
SIP	Session Initiation Protocol
SiPp	SIP protocol traffic generator
SLA	Service Level Agreement
SM	Service Model
SMF	Session Management Function
SMO	Service Management and Orchestration
SVMs	Support Vector Machines

TB	Transport Block
TDD	Time-Division Duplexing
tbit/s	Terabits per second
THz	Terahertz
TPR	True Positive Rate
TPU	Tensor Processing Unit
TSF	Time Series Forecasting
UE	User Equipment
UL	Uplink
UP	User Plane
UPF	User Plane Function
URLLC	Ultra-Reliable Low-Latency Communications
XR	Extended Reality

Chapter 1

Introduction

1.1 Evolution towards 6G Networks

The evolution of mobile communications has continuously reshaped global connectivity. Early networks (2G/3G) laid the groundwork for basic voice and data services, while 4G LTE enabled multimedia-rich applications. The advent of 5G further revolutionized connectivity by introducing three broad service categories: enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communications (URLLC), and massive Machine-Type Communications (mMTC) [9]. The eMBB expands capacity and data rates to support bandwidth-intensive applications (e.g. 4K/8K video streaming, mobile broadband access), with 5G targets of up to 20Gbps peak download rates. URLLC focuses on mission-critical services by providing end-to-end latencies on the order of 1ms and extremely high reliability (99.999%), enabling applications like industrial automation, vehicle-to-X communication, and remote surgery. mMTC supports massive IoT deployment, connecting up to 10^6 devices per square kilometer with improved energy efficiency for sensor nodes.

Despite its advances, 5G still faces fundamental challenges in meeting emerging requirements. One issue is scalability – handling an ever-growing density of devices and data traffic while maintaining performance. For instance, 5G's design goal of one million devices per km^2 may be insufficient for the "Internet of Everything" envisaged in the coming decade. Another concern is energy efficiency: dense 5G deployments (especially mmWave cells and massive MIMO antennas) can consume considerable power, and battery-powered IoT devices still struggle with limited lifespans. Future networks call for an order-of-magnitude improvement in energy efficiency. Additionally, delivering deterministic ultra-low latency is

challenging with 5G's current architecture – while 1 ms radio link latency is possible in ideal cases, guaranteeing consistently low end-to-end latency for time-sensitive applications like industrial control remains difficult. Techniques like Time-Sensitive Networking integration are only partial solutions in 5G. Furthermore, 5G networks are not inherently "intelligent" or context-aware; they primarily react to network conditions rather than proactively adapting to the semantic needs of applications.

6G is envisioned to address these gaps by design. Initial discussions project 6G deployment around 2030 [10] and research and development efforts have already commenced worldwide with several flagship initiatives. These initiatives underscore a global consensus that 6G will be a transformative leap rather than an incremental upgrade. Indeed, early 6G vision documents agree on certain key differentiators that will set 6G apart from 5G:

- **AI-Native Infrastructure:** 6G networks are expected to be designed from the ground up with artificial intelligence and machine learning deeply integrated into control and management planes. Whereas 5G added some AI-driven features as add-on solutions, 6G will be "AI-native", enabling fully autonomous network operations "with zero human touch" [11]. This means tasks like resource allocation, fault detection, and optimization of radio parameters could be handled by AI agents in real time across distributed network elements. An AI-native 6G core would allow the network to learn and self-optimize end-to-end. This is critical for handling complexity: 6G must coordinate many more antennas, nodes, frequency bands, and service types than 5G. By embedding AI at its core, the network can flexibly manage traffic and slice resources far more efficiently than fixed algorithms. Native AI also contributes to sustainability (by continually finding energy-saving strategies) and to resilience (by predicting and mitigating faults or security threats).

- **Integrated Sensing and Communication (ISAC):** Beyond just moving data, 6G will also sense and map the environment. This concept, also known as Joint Communication and Sensing (JCAS), treats radio signals as a tool for situational awareness in parallel with information transfer. This means that 6G base stations and devices perform radar-like functions: measuring reflections of radio waves to detect objects, track motion, and localize targets with extreme precision.

- **Use of Terahertz Bands:** To achieve a Terabit-per-second throughput performance,

6G will expand into previously untapped spectrum, specifically the sub-terahertz range (0.1–1 THz). 5G New Radio reaches up to millimeter-wave frequencies (24–52 GHz, with experimental use up to 100 GHz), but 6G is aiming at using the higher frequency range in the upper mmWave and THz band. These frequencies bring enormous raw bandwidth of tens of GHz per channel, enabling data rates ranging from 100 Gbps to 1 Tbps [12]. Terahertz waves are also very directional and short-range, which aligns with ultra-dense networks of the future where cells might cover small areas or specific hotspots. However, THz propagation faces high free-space path loss and susceptibility to blockage by obstacles. This drives 6G research into advanced wireless transport technologies – for example, novel ultra-massive MIMO antenna arrays and reflecting intelligent surfaces (RIS) to redirect signals.

- **Sustainability and Energy Efficiency:** A core design tenet for 6G is sustainable networking, reflecting both environmental concerns and practical operational costs. This manifests in multiple ways in 6G. First, network hardware and deployments should be far more energy-efficient (bits per joule) than today; 6G targets suggest a 10-fold improvement in network energy efficiency relative to 5G, achieved through techniques like energy-aware scheduling, adaptive sleep modes for network nodes, and use of AI to minimize power use during low traffic [13].

- **Precision Localization and Navigation:** With its new sensing abilities and wide bandwidth signals, 6G will offer localization services of unprecedented precision. The goal is to locate devices or objects to within a few centimeters or even millimeters. In 6G, the high-frequency waveforms and large antenna arrays can be exploited for techniques like angle-of-arrival and time-difference-of-arrival estimation with extreme accuracy.

The Fig. 1.1 illustrates the key areas of 5G upon which the researchers of the flagship 6G project, Hexa-X, will build to develop future 6G technologies. Additionally, Fig. 1.2 from Huawei emphasizes the integration of new technologies such as AI, sensing, and enhanced mobile broadband (eMBB+) in the AI-centric paradigm envisioned for 6G networks. To summarize the technological leap from 5G to 6G, Table 1.1 provides a high-level comparison of their performance targets and representative applications. In every critical dimension—bandwidth, latency, density, intelligence—6G aims for an order-of-magnitude improvement over 5G, facilitating a new era of applications closely integrating our physical and digital

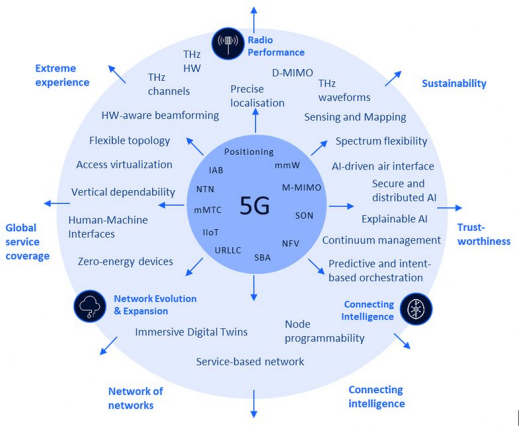


Figure 1.1: Hexa-X 6G research focus areas building upon 5G [14]

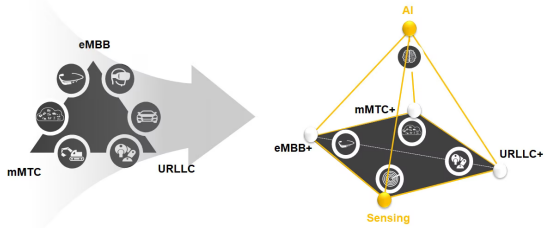


Figure 1.2: Huawei's vision on 6G: from 5G to AI-centric 6G [15]

experiences.

The general context of 6G networks is one of convergence: communications technology converging with differemt domains (AI, control systems, sensing, energy efficiency), and previously separate network types (3GPP, non-3GPP, terrestrial cellular, satellite, short-range device links) converging into one fabric. This holistic vision is driving academia and industry to collaborate on defining 6G's requirements and technologies today, so that a decade from now, society can begin taking advantage of the benefits of the future applications. However, realizing this transformative potential involves overcoming critical technological and operational challenges. The next section dives into some of these challenges in detail, highlighting key motivating factors that drove this thesis such as the underpin the need for dynamic resource allocation, intelligent network management, and robust security mechanisms in the evolution toward 6G.

1.2 Management and Operation Challenges for 6G Networks

The ambitious vision of 6G, introduces some architectural and algorithmic challenges. The requirement of seamless service continuity implies that users and devices must experience uninterrupted, context-aware services even as they move across heterogeneous networks or switch between connectivity modes. Achieving this is non-trivial; 6G will need to coordinate across multiple radio access technologies and network domains to maintain sessions without degradation. Similarly, the demand for extreme performance in terms of latency and reliability forces a continuous optimization of resource allocation algorithms. Network re-

Table 1.1: Comparison of 5G vs. 6G Performance Metrics and Applications

Metric/Feature	**5G (IMT-2020)**	**6G (IMT-2030 Targets)**
Peak Data Rate	$\sim$20 Gbps	$>$1 Tbps
User Experienced Rate	$\geq$100 Mbps	$\geq$1 Gbps
Latency (Air Interface)	$\leq$1 ms	$\leq$0.1 ms
Reliability (BLER)	99.999% (5-nines)	99.99999% (7-nines)
Connection Density	10^6 devices/km^2	10^7 devices/km^2
Mobility Support	Up to 500 km/h	Up to 1000 km/h
Spectrum Bands	Sub-6 GHz, mmWave ($<$100 GHz)	mmWave, sub-THz, optical
Network Architecture	Service-based, network slicing	AI-native, dynamic slicing
Representative Use Cases	eMBB (AR/VR, HD media), URLLC (V2X, industrial IoT), mMTC (IoT)	XR/holography, tactile Internet, autonomous systems, AI-driven applications

sources like spectrum, antenna layers, computing and storage capacity must be allocated in a far more dynamic and fine-grained manner than in 5G, since static or coarse resource slicing would be unable to meet the highly variable Quality-of-Service requirements. This is further complicated by mobility: 6G is expected to serve highly mobile scenarios (e.g. high-speed transport, drones, satellites) while still guaranteeing ultra-low latency and high reliability. Maintaining service quality for moving users may require predictive handover strategies and on-the-fly reconfiguration of network paths or function placement, straining existing mobility management protocols. Meanwhile, the security issues for 6G become even more complex. The network's core design is expected to be cloud-native, fully relying on softwarized—functions that once ran on dedicated hardware are virtualized. This opens new attack surfaces and failure modes, mainly towards exploits of virtualized network functions. In other words, the extreme performance targets, the distributed intelligence, and the software-defined

flexibility open new challenges to continuous service delivery, resource optimization, mobility management, and secure operation. One key enabler to meet 6G's low-latency and high-bandwidth goals is the integration of edge computing deeply into the network architecture. By deploying computational workloads and services at Multi-access Edge Computing (MEC) servers closer to end-users, 6G can dramatically reduce communication delays for applications such as immersive augmented reality or real-time control. Pushing intelligence and content to the edge is seen as crucial for handling massive IoT, Industry 4.0, and other data-intensive use cases while avoiding backhaul bottlenecks [16]. However, this distributed service scheme raises the problem of orchestration under mobility. As users or devices move, the network may need to migrate an ongoing service (e.g. a video analytics application or a VR rendering engine) from one edge node to another that is closer to the user's new location, in order to meet the latency and bandwidth needs. Such service migration must occur seamlessly and without user-perceived interruption. Orchestrating this in real time is extremely challenging: it requires predictive analytics to determine when to migrate, selection of an optimal target edge node with available resources, state transfer and synchronization between the source and target, and careful handling of the underlying connectivity handover so that the session persists. In 6G, with its more strict continuity requirements, new mechanisms are needed to handle live state transfer and indirection of traffic on the fly. The complexity is exacerbated in real-world deployments where multiple infrastructure providers and domains are involved (e.g. a user moving from one operator's coverage to another's, or from an indoor private network to a macro network). In essence, edge computing will be crucial in 6G, but it transforms the mobility management problem into a joint communication-computation and orchestration problem.

Another emerging challenge lies in managing dynamic resource contention in dense radio environments. 5G already introduced the concept of network slicing to isolate and guarantee resources for different services (e.g. an eMBB slice for broadband streaming, an URLLC slice for mission-critical control). In 6G, slices will need to become more adaptive and "service-aware." The network is expected to support an even wider variety of application types simultaneously – from ultra-low-latency industrial control to bandwidth-hungry holographic displays – often over the same physical infrastructure. In dense deployments with many base stations and devices (including new types like IoT sensors, vehicles, and drones), the interference and load conditions can fluctuate rapidly. A static or one-time slice configuration would

either frequently violate the service requirements or waste capacity. Thus, there is a strong motivation for predictive, AI-driven slicing in 6G. The idea is to continuously monitor network conditions such as traffic patterns, radio quality, user behavior and anticipate the future demand. Machine learning models can forecast, and the network controller could proactively adjust slice allocations or scheduling priorities in advance. By being service-aware, the slicing mechanism would also consider the specific QoS needs of each application type – for instance, giving transient bursts of extra radio resources to a vehicular communication slice when a safety message is detected, or temporarily reduce the throughput of a delay-tolerant slice to ensure a VR stream remains within its latency range. Achieving this level of agility requires advances in RAN algorithms and cross-layer optimization. It also demands data collection from the network, and closed-loop control to enforce the slice adjustments at runtime. Early research anticipates that incorporating AI at the RAN scheduler and slice orchestrator will be a necessity for 6G, enabling the network to learn and auto-tune its resource allocation policies on the fly [17]. We also must recognize that user behavior and radio conditions are inherently non-stationary, fluctuating over time in ways that cannot be fully generalized. This means that 6G networks must employ online learning and AI-driven optimization at scale. In contrast to traditional networks where algorithm parameters are set through offline planning or human tuning, an AI-native 6G network would continuously learn from real-time data. For example, the network can observe patterns of user mobility, application usage peaks, or signal quality variations, and gradually improve its predictive models for traffic and channel conditions. These models can then drive decisions such as routing, handover timing, power control, and cache placement with greater accuracy than static heuristics. To support such capabilities, the network architecture must integrate a distributed machine learning pipeline – often referred to as network MLOps (Machine Learning Operations). This includes mechanisms for data collection and curation (from base stations, user devices, core network elements), training or updating models (which might occur in centralized cloud nodes or hierarchically across edge and central nodes), and deploying the updated models back into the network elements for inference. Frameworks for closed-loop automation, such as ETSI's Experiential Networked Intelligence (ENI) [18] or the 3GPP Network Data Analytics Function (NWDAF) introduced in 5G, provide early examples of how analytics and learning can be looped into management decisions.

The challenge is to design these learning loops such that they converge quickly, remain

stable, and make beneficial decisions to network performance. Additionally, the overhead of data shipping and model training must be controlled so as not to consume excessive network resources. Despite these challenges, the benefit is significant: a network that improves with experience, adapting to new conditions or services in a matter of minutes or hours instead of requiring months of re-engineering. But with the autonomy of 6G comes a great concern for security and resilience. There is a recognized need for intelligent intrusion detection and mitigation systems that operate hand-in-hand with the orchestration and control plane. 6G networks will likely employ AI-driven security monitors that learn baseline behavior and can detect anomalies in traffic patterns or network state in real time. For example, a distributed Intrusion Detection System (IDS) could analyze signals from many parts of the network (CPU usage spikes, unusual signaling sequences, drops in QoS) and correlate them to flag a coordinated attack on a network slice or a set of base stations. Once an intrusion or anomaly is detected, the network should be able to automatically respond – this could mean isolating portions of the network, re-routing critical services to safe resources, pushing software patches, or re-training an affected AI model on the fly to exclude tainted data. Such adaptive security management blurs the line between networking and security operations (NetOps and SecOps), pointing toward a future of autonomous security orchestration. The challenge is that these countermeasures themselves must act within the strict latency and reliability bounds of 6G services.

To tackle the above challenges, the networking industry is converging on new architectural frameworks that bring intelligence and openness into network management. Such an example is the O-RAN (Open RAN) architecture, which is expected to play a central role in next-generation RAN management. O-RAN decouples the RAN into open, interoperable components and introduces the RAN Intelligent Controller (RIC). There are two types of RIC in O-RAN: the non-real-time RIC (within the service management and orchestration layer), which handles tasks on the order of seconds or longer (e.g. policy generation, model training), and the near-real-time RIC (at the edge of the RAN), which can execute control loops with latency on the order of 10ms to 1s. Developers can write xApps (for the near-RT RIC) and rApps (for the non-RT RIC) that implement custom control logic—ranging from smarter handover algorithms, to predictive scheduling, to anomaly detection in the RAN. This opens the RAN to programmability and innovation in a way not possible in previous generations.

In conclusion, the vision of 6G as an autonomously orchestrated, AI-powered, edge-to-

cloud computing fabric brings many research challenges. These span continuous service continuity under mobility, agile resource allocation and slicing in dynamic conditions, continual learning and adaptation, and embedding security and trust into every layer of a softwarized architecture. Addressing these challenges will require innovations in network architecture, algorithms, and protocols. The telecom industry's early efforts such as edge computing advances, AI-native network prototypes and O-RAN standardization are laying the groundwork, but many open questions remain. This thesis, in the following chapters, will build upon this context, focusing specifically on a subset of these emerging problems, and propose novel contributions toward enabling intelligent management of 6G networks.

1.3 Thesis Contributions to 6G Management and Operations

In light of the previously outlined challenges, this thesis sets out to investigate how 6G networks can achieve intelligent, adaptive, and secure management through AI-driven mechanisms and cloud-native orchestration. The underlying question motivating this research is:

> *How can end-to-end Quality of Service (QoS) guarantees be delivered dynamically in 6G networks, especially under conditions of user mobility, resource contention, and potential security threats?*

To answer this, the thesis addresses the problem across three key dimensions: **service continuity through edge computing**, **adaptive resource slicing**, and **resilient AI-native network security**.

1. Seamless Service Continuity via Edge Computing and DRL-based Migration

Research Questions:

- *How can MEC services be dynamically migrated in real-world networks to follow user mobility and maintain low-latency performance?*
- *How can the network anticipate user movement and proactively adapt computational and networking resources?*

- *How can we evaluate the performance of the proposed migration framework in realistic scenarios?*

Contribution: In chapters 3, 4 we designed, implemented, and evaluated a cloud-native Follow-Me MEC architecture based on disaggregated and heterogeneous base stations and containerized edge services. Through extensive experimentation on a testbed, we demonstrated seamless low-latency MEC service continuity. To optimize migration decisions, we proposed a Deep Reinforcement Learning (DRL) controller that anticipates mobility by learning multi-cell latency patterns and server load, and dynamically selects the optimal target edge. Our approach significantly outperforms naive or reactive strategies in terms of service downtime and user-perceived latency.

2. Service-Aware Resource Slicing through Deep Learning and MLOps

Research Questions:

- *How can network slices be dynamically adapted to the evolving demands of mobile users and applications?*
- *How can user behavior and radio condition forecasts be leveraged to preemptively allocate resources?*
- *How can we deploy and update ML models efficiently in real-world, cloud-native network environments?*

Contributions: In chapter 5, the thesis introduces a service-aware slicing framework that integrates time-series forecasting models trained on traffic, QoS, and channel quality indicators (CQI). By using deep models such as LSTM, GRU, and Bi-LSTM, we accurately predicted future demand trends and adapted RAN slices proactively via programmable APIs. To maintain accuracy in real-time deployments, we developed a full MLOps pipeline capable of online and distributed training, model monitoring, and inference within a Kubernetes-managed infrastructure. This platform enables continual adaptation to user and channel dynamics, addressing the non-stationarity inherent in 6G usage scenarios. Our findings indicate that the network can swiftly adjust to traffic, providing users with slices tailored to their application needs. Notably, our experiments show that under the studied settings, the users experienced up to 4 times lower latency (jitter) and nearly 4 times higher throughput when interacting with various applications, compared to the standard non-AI/ML unit.

3. Secure and Intelligent Resource Control via Network Intrusion Detection and xApp Orchestration

Research Questions:

- *How can 6G networks detect and react to malicious traffic patterns that impact QoS?*
- *How can AI-based intrusion detection be integrated with network control in an open, softwarized RAN?*

Contributions: In chapter 6 we designed a network anomaly detection module capable of identifying abnormal traffic flows using Random Forest, SVMs, and Autoencoder models trained on real packet statistics. This module was integrated into the Open RAN architecture using the near-RT RIC, where an xApp continuously monitors per-user flows and triggers policy changes such as rellocation of Radio Resource Blocks and user management upon detecting threats. This closed-loop system ensures that QoS for legitimate users remains intact even under attack, and paves the way for autonomous security enforcement in future RAN deployments. Experimental evaluations show that our system effectively maintains low latency under attack conditions, nearly doubles the throughput for legitimate users, and reduces average CPU usage by up to 15%.

The cumulative result of this work is a modular, AI-native orchestration stack for future wireless networks. Each component—from service migration at the edge, to predictive RAN slicing, to anomaly-aware policy enforcement—has been validated through experimental evaluations in testbed environments with realistic traffic and mobility patterns. Together, these contributions offer a practical roadmap for building intelligent and resilient 6G infrastructures.

1.4 Other Research Contributions (Out of Scope of This Thesis)

In parallel to the main contributions of this thesis, the author also actively participated in side projects and tool development that, while not forming part of the core thesis objectives, demonstrate valuable technical expertise and relevance to the broader 6G research ecosystem. Bellow are some of the notable contributions:

SLICES-RI:

SLICES Research Infrastructure (SLICES-RI) project, is a European initiative aimed at providing a flexible, multi-site, and programmable experimentation platform for advancing research in 5G, post-5G, and 6G technologies. Further details on SLICES-RI can be found in chapter 2.4. The author contributed to the SLICES-RI project in the following ways:

- Multi-Cluster Provisioner: The author designed and implemented a custom Kubernetes cluster provisioner capable of dynamically creating and managing multiple experimental clusters. This tool leverages KubeVirt to deploy virtual machines as Kubernetes nodes and uses Rancher APIs to register and configure them as part of managed clusters. The provisioner automates the entire lifecycle, from VM instantiation to cluster registration, facilitating repeatable, isolated experimentation environments for different user groups or research objectives. The system supports integration of both RKE2 and K3s distributions and introduces options for advanced network configurations (e.g., Multus, DNS, TLS-SANs).

- Integration of a RIC in the Post-5G Blueprint: As part of the post-5G blueprint maintained within SLICES-RI, the author contributed to the integration of the FlexRIC RIC Controller. FlexRIC provides a modular and programmable framework to control the RAN via standardized near-real-time RIC interfaces. The author's work involved adapting the control interface logic and ensuring the FlexRIC instance could communicate with emulated or real RAN components in a containerized testbed.

- Automated Provisioning of a USRP-Based RAN Installation: Automation of a USRP-based RAN setup, including the deployment of OpenAirInterface (OAI) gNB instances and associated CN components. The automation framework configures the USRP hardware, synchronizes RF parameters, and provisions the RAN stack onto compute nodes via Ansbile. This reduces the manual configuration overhead and enables rapid deployment of real-world RANs for experimentation within the post-5G blueprint.

OpenAirInterface & FlexRIC:

The author has been actively involved in the OpenAirInterface project, an open-source initiative that provides comprehensive platforms and tools for 5G and beyond wireless research and experimentation. Detailed information about OAI and FlexRIC tools utilized throughout this thesis is provided in Chapter 2.4. In collaboration with the OAI and FlexRIC

communities, the author contributed within OAI RAN implementations with standardized O-RAN specifications. Specifically, the contributions involved developing and integrating standardized O-RAN Type 1 RAN Control (RC) reporting mechanisms, enhancing OAI's capabilities to meet O-RAN Alliance standards. Due to the current limitations within OAI's RAN implementation—particularly incomplete support for certain handover events involving NG or Xn interfaces—the current implementation focuses primarily on UE attachment-related events. Nevertheless, these initial contributions establish the groundwork for broader support of a comprehensive range of RRC-triggered events within OAI and FlexRIC. Key contributions integrated into the FlexRIC and OAI platforms, are summarized as follows:

- Extension of the RC monitoring xApp: In order to support all standardized RC REPORT Styles (1 to 5). This advancement significantly enhances FlexRIC's capability to monitor and control radio resources at granular levels.
- Encoding/Decoding RRC Messages: Enables detection and reporting of protocol messages such as RRC Setup Complete, RRC Reconfiguration, Security Mode Complete, and Measurement Report.
- UE ID: Allows tracking of unique UE identifiers based on triggering events like the RRC Setup Complete or F1 UE Context Setup Request.

The modifications have been integrated upstream into both the FlexRIC and OpenAirInterface codebases.[1][2]

1.5 Thesis Structure

This thesis is structured into seven main chapters, following a logical progression from context and background to proposed solutions and evaluation:

- **Chapter 1 – Introduction:** Presents the motivation behind this research, the evolution towards 6G networks, the management and operation challenges, and outlines the thesis contributions, including a brief overview of additional research activities.

[1] FlexRIC commit: `https://gitlab.eurecom.fr/mosaic5g/flexric/-/commit/3690c54e9954e7fc1af98a1cdf3e6dc179998db0`

[2] OpenAirInterface commit: `https://gitlab.eurecom.fr/oai/openairinterface5g/-/commit/e6797a0b0e80eba13d51e738e4fa90df9f763b32`

- **Chapter 2 – Background:** Provides essential background on 5G/NR architecture and slicing mechanisms, Multi-access Edge Computing (MEC), and introduces key concepts in Artificial Intelligence and Machine Learning, including tools and testbeds used throughout the thesis.

- **Chapter 3 – Mobility-Aware Edge Service Migration for 6G Networks:** Introduces the follow-me MEC concept and describes a lightweight architecture for edge service migration under user mobility constraints. Evaluation results on real testbeds demonstrate the feasibility of seamless low-latency handovers.

- **Chapter 4 – Deep Reinforcement Learning based Service Migration for 6G Networks:** Extends the service migration framework using a Deep Reinforcement Learning (DRL) agent to optimize decision-making in dynamic and resource-constrained edge environments.

- **Chapter 5 – Service-Aware Network Slicing for 6G Networks:** Proposes a dynamic slicing mechanism powered by time-series deep learning models. The chapter includes the architecture of an AI/ML forecasting unit and a Kubernetes-based MLOps stack for model training, inference, and deployment.

- **Chapter 6 – AI-Driven Attack Mitigation using Slicing for 6G Networks:** Addresses the issue of security in 6G RANs. It presents a network anomaly detection system using classical ML algorithms and autoencoders, integrated with slicing and user management logic via an xApp to enforce policy actions in real time.

- **Chapter 7 – Conclusions:** Summarizes the main findings, reflects on the contributions, and outlines future directions for extending this research, including deployment in live RIC platforms and scaling AI pipelines.

Supporting materials such as figures, tables, abbreviations, and publications are included in the beginning of the thesis for easier readability and reference. A complete bibliography is provided at the end.

Chapter 2

Background

This chapter presents the fundamental technologies, methodologies, and tools extensively studied throughout this research. We begin by introducing the core concepts of 5G/NR networks in Section 2.1, followed by a discussion of Software Defined Networks (SDN) and their integration into 5G networks within the O-RAN architecture in Section 2.1.6. Section 2.2 explores the Multiple Access Edge Computing (MEC) paradigm, emphasizing its role in enabling low-latency applications. We further introduce AI/ML architectures and models in Section 2.3 that we relied in order to enhance resource optimization in RAN and edge environments. Finally, Section 2.4 provides an overview of the tools and testbeds used to evaluate the proposed solutions, which are detailed in the subsequent chapters.

2.1 5G/NR

2.1.1 Architecture Overview

The 5G network architecture is designed with two primary configurations: Non-Standalone (NSA) and Standalone (SA) as illustrated in Fig. 2.1. In the NSA configuration, the 5G Radio Access Network (RAN) is integrated with the 4G core, providing a limited set of 5G capabilities. Both LTE and 5G RAN components connect to a shared core network, enabling communication through the Xn interface.

In contrast, the SA configuration is fully independent, with the 5G RAN supported exclusively by the 5G core network. The core is built using a Service-Based Architecture (SBA), designed to be cloud-native and fully softwarized. This structure provides increased flexibility and scalability for deploying network functions. The 5G RAN uses New Radio (NR)

antennas, known as gNodeBs, replacing the LTE's eNodeB antennas and allowing for more advanced capabilities.

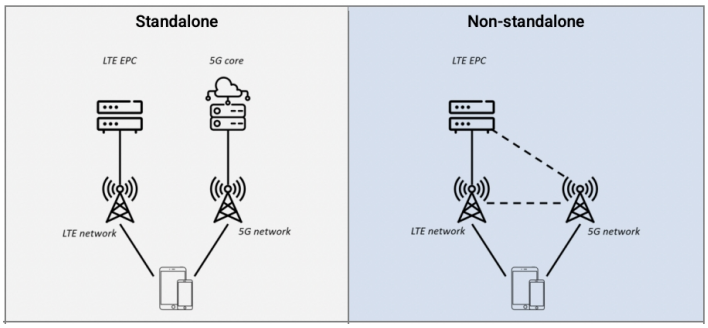


Figure 2.1: SA and NSA Architectures

Going deeper into the 5G architecture Fig. 2.2 shows the comprehensive, end-to-end layout of a 5G SA network. It is structured into two main components: the Core Network (CN), also known as the 5G Core (5GC), and the Radio Access Network (RAN), often referred to as New Radio (NR). The RAN includes base stations, known as next-generation Node B (gNB), and User equipment (UEs). Each gNB is designed to handle all radio-based functions, establishing and maintaining wireless connectivity for UEs. These gNBs are interlinked through the Xn interface, facilitating coordination for processes like handover when a UE moves between cells, thus ensuring seamless connectivity.

The SBA-core network offers more flexibility and opportunities for network enhancements. Such innovation enables the network operators to deploy the network functions as microservices rather than depending on hardware. Furthermore, there is a distinct Control and User plane separation (CUPS). This approach allows network functions to operate with flexibility either as dedicated components or as shared resources across different network slices/regions, depending on specific service requirements.

The Access and Mobility Management Function (AMF) serves as the control anchor from the RAN's perspective, managing essential control signaling operations like registration and mobility support between the CN and UEs. The Session Management Function (SMF) provides IP addresses to UEs and manages their session. Additionally, other key functions include the Policy Control Function (PCF) for policy enforcement, Unified Data Management (UDM) for secure user data storage, the Authentication Server Function (AUSF) to ensure

secure authentication, and the Network Slice Selection Function (NSSF), which selects the most appropriate slice the end-users. Since everything operates as a service, the Network Repository Function (NRF) simplifies network management by performing service discovery, and the Network Exposure Function (NEF) simply exposures the service to/from 3rd party applications.

On the user plane, the primary function is handled by the User Plane Function (UPF), which facilitates data transfer between the RAN and external networks, such as the Internet. The UPF performs several tasks, including routing, packet inspection, and QoS control. To achieve differentiated packet handling, each user can have one or more Protocol Data Unit (PDU) sessions, which consist of QoS flows for individual applications or services. In the core network, IP flows are aligned with QoS flows and tagged to convey service requirements, ensuring network slices can deliver precise, context-aware services. Within the RAN, these QoS flows correspond to data radio bearers, which are essential for RAN-level communication between the gNB and UE.

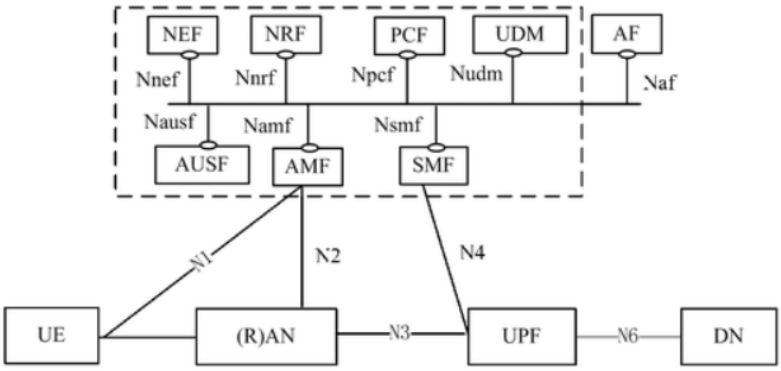


Figure 2.2: 5GNR architecture

In comparison, the 4G architecture illustrated in Fig. 2.3, known as the Evolved Packet System (EPS), shares a similar structural setup but with notable differences. The EPS includes LTE for the RAN component and the Evolved Packet Core (EPC) as its core. Here, the base stations are termed evolved Node B (eNB), and the interconnection between eNBs is facilitated via the X2 interface. In the EPC, in contrast with 5GC, there is a hardware dependency on network functions. The Mobility Management Entity (MME) is the control plane anchor, connected to eNBs through the S1-c interface, with subscription information maintained by

the Home Subscriber Server (HSS). On the user plane, the Serving Gateway (SGW) anchors the data path for eNBs, utilizing the S1-u interface for data transmission.

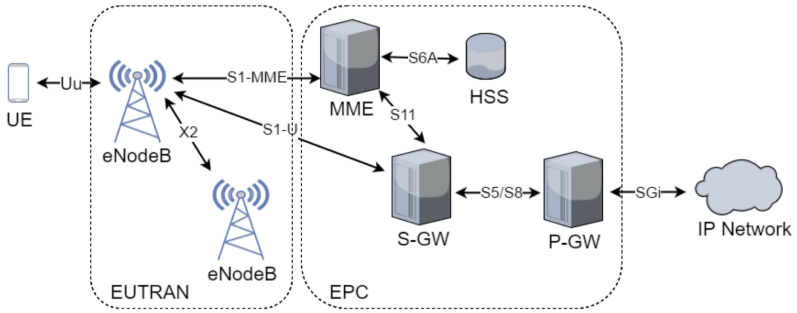


Figure 2.3: LTE architecture

2.1.2 RAN Protocol Stack

The RAN protocol stack plays a crucial role in managing the interface UEs and the core network. Furthermore, the protocol stack is responsible for Mobility Management, Resource Allocation, Error Correction, and QoS Optimization. The RAN extends the CUPS logic into the RAN protocol stack. Specifically, the RAN protocol stack is divided into protocols responsible for the user plane and protocols that manage the signaling traffic and control plane. In Fig. 2.4 we can observe the different Protocol stacks for both UE and gNB, which almost have identical stacks but with different responsibilities that vary due to the gNB's control over the UEs in the network. For example, the stack in each gNB and UE contains layers for Physical (PHY), Medium Access Control (MAC), Radio Link Control (RLC), Packet Data Convergence Protocol (PDCP), Radio Resource Control (RRC), and Non-Access Stratum (NAS). However, due to the gNB's role in controlling network operations, each layer on the gNB side carries additional responsibilities, particularly within the CP. Bellow we analyze the different functionalities of each protocol:

- **PHY**: At the base of the stack, the PHY layer encodes and modulates data for transmission over the air. It manages the fundamental signal processing tasks like encoding, modulation, and antenna mapping.

- **MAC**: The MAC layer sits above the PHY and coordinates access to the shared radio resources among UEs. Responsibilities include scheduling, error correction (using Hybrid Automatic Repeat Request), and QoS management. Also multiplexes each UE's data into transport blocks. The scheduler within the MAC layer allocates resources dynamically.

- **RLC**: The RLC layer receives data from the upper layer PDCP and stores them in buffers until they can be transmitted over the air. This layer segments data packets when necessary and can also perform retransmissions if set to Acknowledged Mode (AM). When configured in Unacknowledged Mode (UM), it transmits data without retransmissions, prioritizing speed over reliability.

- **PDCP**: The PDCP layer handles both CP and UP tasks. For UP it provides retransmission and reordering through sequence numbering, header compression, and encryption. The CP applies integrity protection, reducing packet overhead, and ensuring efficient and secured data transmissions.

- **RRC**: The RRC layer orchestrates the connection setup and mobility management between UEs and the network. But interfacing with AMF it mainly maintains the UE's state, and manages handover decisions, and security configurations ensuring seamless connectivity as users move across different cells or network regions.

- **NAS**: The NAS protocol is responsible for connecting the UE and AMF, applying network security algorithms.

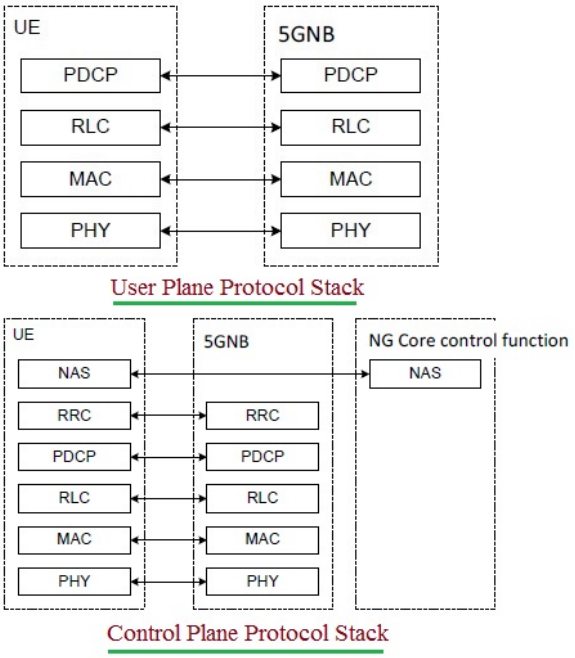


Figure 2.4: 5G Radio Protocol Stack

2.1.3 RAN Functional Splits

To achieve greater flexibility, scalability, and efficiency in network deployments, the 3rd Generation Partnership Project (3GPP), divided gNodeB into distinct functional units. 3GPP protocol stack divides functions between the Centralized Unit (CU) the Distributed Unit (DU) and the Radio Unit (RU) as illustrated in Fig. 2.5. This disaggregation supports virtualizing functions on commercial, off-the-shelf hardware, providing operators with adaptable and cost-effective deployment options. As summarized in the previous subsection, the RAN protocol stack's main layers include Layer 1 (PHY), Layer 2 (MAC, RLC, PDCP), and Layer 3 (RRC, NAS). In the disaggregated architecture the CU supports the PDCP and RRC while the DU is responsible for real-time scheduling, and aggregates the lower layers including RLC MAC and PHY. The PHY layer through specific option splits can be supported only through RU. It's important to mention that CU can be further separated into CU-User Plane CU-UP and CU-Control Plane CU-CP for the user and control functionalities respectively. The CU-CP manages signaling and control functions, using protocols like RRC for connection and mobility management and PDCP for integrity protection. The CU-UP, on the other hand, handles user data, employing PDCP for secure data transfer and QoS mapping. This disaggregated structure not only enables enhanced resource allocation and network customization but also supports the integration of edge computing by placing DUs/CU-UPs nearer to users.

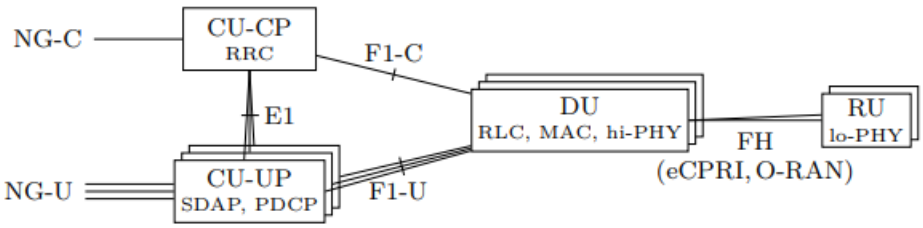


Figure 2.5: Disaggregated 5GNR Architecture

Depending on deployment needs 3GPP defined different Functional Split options in Technical Requirement (TR) 38.801 [19]. The Key Split Options are demonstrated in Fig. 2.6 and are the following:

- Split Option 1: Places RRC in the CU while keeping other functions within the DU and RU.
- Split Option 2: Moves both RRC and PDCP to the CU, leaving lower layers in the DU and RU.
- Split Option 3 (Intra-RLC): RF, PHY, MAC, and low RLC are in the DU/RU, while higher functions remain centralized.
- Split Option 4: MAC, PHY, and RF are distributed, while RLC, PDCP, and RRC are in the CU.
- Split Option 5 (Intra-MAC): RF, PHY, and certain MAC functions (e.g., HARQ) are in DU/RU, with upper layers centralized.
- Split Option 6: RF and PHY are in the DU/RU, while upper layers are in the CU.
- Split Option 7 (Intra-PHY): Splits the physical layer, placing some PHY functions and RF in the DU/RU and others in the CU.
- Split Option 8: Centralizes all functions except RF within the CU.

Lower-layer splits, close to RF, require higher bandwidth and stricter latency but increase scalability as antenna ports increase. Higher splits, such as those near the PDCP layer, reduce bandwidth needs, relax latency requirements, and allow CU-RU connections across greater

distances [20]. In LTE, baseband processing functions (e.g., RRC, PDCP, RLC, MAC, PHY) are handled by a Base Band Unit (BBU), while the RF functions reside in the Remote Radio Head (RRH). In contrast, a common 5G implementation is split 7, with low PHY and RF in the RU, and upper functions in a CU/DU setup.

The connection between CU-CP and CU-UP utilizes the E1 interface, standardized by 3GPP and operating with split option 1 for efficient control and user data communication. One CU-CP can manage multiple CU-UP instances scaled based on network demand. The F1 interface links the CU to the DU, using option 2, facilitating centralized control and distributed processing. In addition, a single CU can manage multiple DUs, supporting scalability and load distribution throughout the network, essential to optimize resources in high-density areas. The DU connects to the RU through various fronthaul interfaces, including nFAPI [21] (option 6), eCPRI (option 7), or O-RAN's open fronthaul [22] (option 7-2x), allowing flexible, vendor-neutral deployments.

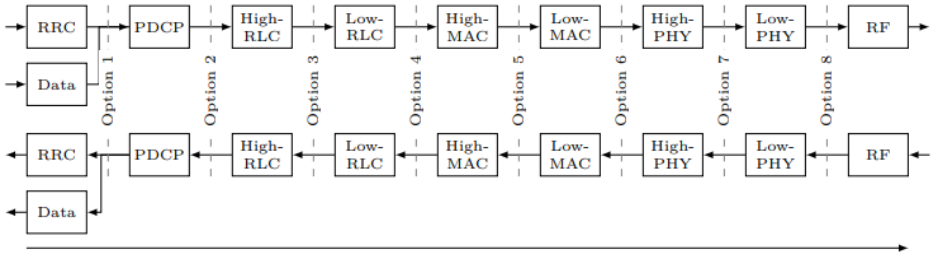


Figure 2.6: Split options in the Disaggregated 5GNR Architecture

2.1.4 RAN Resource Allocation/Slicing

Both LTE and 5GNR widely adopt a multicarrier modulation technique named Orthogonal Frequency-Division Multiplexing (OFDM), depicted in Fig. 2.7. It divides the available spectrum into numerous orthogonal subcarriers, each modulated with a low-rate data stream. This orthogonality ensures that subcarriers do not interfere with each other despite overlapping in the frequency domain. Such signal can be expressed mathematically by the following Eqn. 2.1, where X_k is the symbol transmitted on the k-th subcarrier, f_k is the frequency of the k-th subcarrier and N is the total number of subcarriers. For sending/receiving such signals both senders/receivers apply Inverse Fast Fourier Transform (IFFT) and Fast Fourier Transform (FFT) respectively to modulate/demodulate frequency-domain data symbols into

a composite time-domain signal and vice-versa. Traditionally, LTE systems employed a fixed subcarrier spacing of 15 kHz, optimized for sub-6 GHz frequency ranges with favorable propagation characteristics. However, the design of NR extends beyond sub-6 GHz (Frequency Range 1, FR1) to include mmWave frequencies (Frequency Range 2, FR2), which exhibit different propagation behaviors such as higher path loss and increased Doppler effects. To address these challenges, 5G NR introduces the concept of numerology, which allows flexible subcarrier spacing. The subcarrier spacing Δf in NR is $2^{\mu} * 15$ kHz kHz, where μ represents the numerology index. This can be scaled with subcarrier spacing values such as 15, 30, 60, 120, and 240 kHz.

$$s(t) = \sum_{k=0}^{N-1} X_k e^{j2\pi f_k t} \tag{2.1}$$

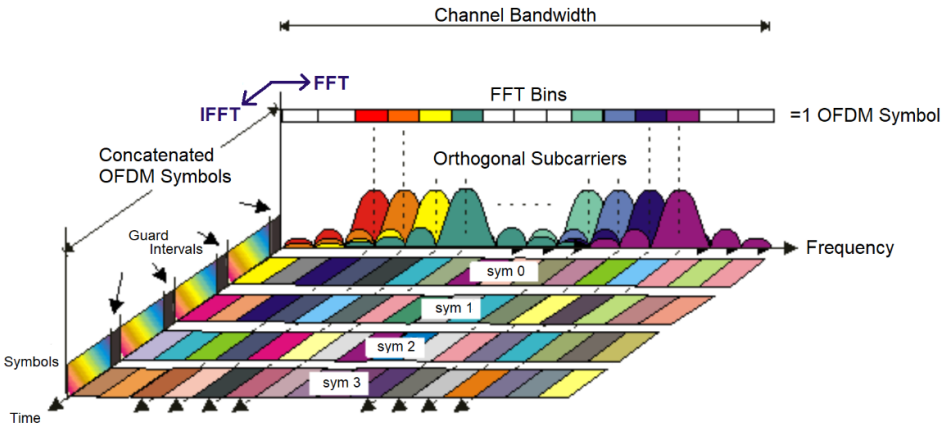


Figure 2.7: OFDM Symbols in Frequency and Time domain.

The smallest allocation unit in the frequency domain is Resource Blocks (RBs). The RBs combine a group of subcarriers into a manageable unit for allocation. For example, in LTE, an RB consists of 12 consecutive subcarriers, each spaced 15 kHz apart, resulting in a total bandwidth of 180 kHz per RB. In 5G New Radio (NR), the subcarrier spacing is scalable based on the numerology index. The number of RBs available within a given bandwidth is determined by the carrier bandwidth and subcarrier spacing. For example, a 100 MHz carrier bandwidth with 15 kHz subcarrier spacing can support approximately 556 RBs. The ability to allocate RBs dynamically across users and services allows the network to optimize spectral efficiency while meeting the QoS demands of different applications.

In the time domain, resources are organized into slots, subframes, and frames. A frame typically spans 10 ms and consists of 10 subframes, each lasting 1 ms. Within each subframe, the time is further divided into multiple slots, the number of which depends on the numerology index. For instance, with a numerology of μ=0 (15 kHz subcarrier spacing), there are two slots per subframe, each 0.5 ms in duration. However, higher numerologies, such as μ=2 (60 kHz subcarrier spacing), result in more slots per subframe (e.g., 8 slots of 0.125 ms each). Higher numerologies in the time domain reduce symbol duration, enabling lower latency and finer scheduling granularity, which is ideal for URLLC applications. However, shorter symbols increase sensitivity to timing errors and inter-symbol interference (ISI). Each time-domain slot is composed of symbols, and the number of symbols per slot is typically 14 for cyclic-prefix OFDM (CP-OFDM).

The combination of frequency domain RBs and time domain slots forms the resource grid, a two-dimensional matrix of time-frequency resources. This grid is the basis for scheduling algorithms, mainly used in the MAC layer through the MAC scheduler. The scheduler mainly allocates resources to users or applications based on factors such as channel quality indicators (CQI), QoS requirements, and traffic demands. There is a different scheduling mechanism for DL and for UL in the scheduler depending on the user requirements, network conditions, and algorithmic designs. The physical allocation of resources is handled at the level of Physical Resource Blocks (PRBs). A PRB corresponds to one RB in the frequency domain and one slot or a subset of symbols within a slot in the time domain. PRBs serve as the building blocks for transmitting control and data information.

While PRBs provide the basic units for allocating spectrum and time resources to users and services, network slicing leverages PRBs to create multiple virtualized networks, or ”slices”, each customized to meet specific application requirements and service demands. Each slice operates as an independent virtual network, optimized for particular types of services such as enhanced Mobile Broadband (eMBB), URLLC, or massive Machine-Type Communications (mMTC). For example, an eMBB slice dedicated to high-definition video streaming can be allocated a contiguous set of PRBs to maximize throughput, while a URLLC slice focused on real-time control applications can be assigned PRBs with minimal latency and high reliability. Network slicing operates on two primary levels: inter-slice and intra-slice slicing. Inter-slice slicing involves the separation of the RAN into completely independent virtual networks (e.g. URLLC, eMBB), each serving a distinct service category. Intra-slice

slicing, on the other hand, refers to the further subdivision of a single network slice into smaller, more specialized segments. Each sub-slice can be allocated a specific subset of PRBs targeted for specific needs. A key element in network slicing is the Network Slice Selection Assistance Information (NSSAI), which enables the network to identify and assign the appropriate slice to a UE during connection. NSSAI has two key components: the Slice/Service Type (SST) and the Slice Differentiator (SD). The SST classifies the slice according to general service categories such as eMBB, URLLC, and mMTC, ensuring that UE connects to a slice optimized for its specific service needs. The optional SD provides finer distinctions within an SST, allowing multiple slices under the same service type. For example, within the eMBB, one SD might support high-definition video streaming while another is for virtual reality applications.

While RBs and PRBs are the foundational structure for time-frequency resource allocation, they operate uniformly across the entire carrier bandwidth. However, as 5G extends into wider frequency ranges and supports heterogeneous services, this one-size-fits-all approach becomes insufficient to efficiently address each use case's unique requirements.

To overcome these limitations, 5G NR introduces Bandwidth Parts (BWPs) to partition the carrier bandwidth into smaller, customizable segments. A BWP is defined as a subset of contiguous resource blocks within the carrier bandwidth, and it operates with specific configurations such as subcarrier spacing, cyclic prefix length, and the number of resource blocks it spans. BWPs are particularly effective in addressing the coexistence of diverse services within a single carrier. For instance, a BWP designed for eMBB may span a large number of RBs to provide the high throughput required for video streaming, while another BWP within the same carrier might cater to URLLC applications, with configurations optimized for low latency and high reliability. This segmentation ensures that resources are allocated precisely where they are needed, avoiding waste and interference.

2.1.5 RAN Dupplexing

Duplexing methods are essential in wireless communication systems to enable two-way communication between UE and the base station. The two primary duplexing techniques employed in RAN are Time Division Duplexing (TDD) and Frequency Division Duplexing (FDD). Each method has distinct operational characteristics, advantages, and challenges that influence its suitability for different deployment scenarios and service requirements.

TDD operates by allocating separate time slots for UL and DL transmissions within the same frequency band. This temporal separation allows for dynamic adjustment of UL and DL resource allocation based on real-time traffic demands. For example, in environments where downlink traffic dominates, such as in video streaming applications, the system can allocate more time slots to DL transmissions, thereby optimizing overall network efficiency. TDD also facilitates easier implementation of advanced techniques like beamforming and Massive MIMO, as the same frequency resources are used for both transmission and reception, simplifying channel state information acquisition. However, TDD presents challenges related to synchronization, especially in heterogeneous network deployments with varying cell sizes and traffic patterns. Precise timing synchronization is critical to prevent interference between cells operating in adjacent time slots. Additionally, TDD systems are more susceptible to issues arising from propagation delays, which can complicate the design of guard periods that separate UL and DL transmissions to mitigate interference.

FDD, on the other hand, assigns distinct frequency bands for uplink and downlink communications. This simultaneous transmission and reception on separate frequency bands enable continuous two-way communication without the need for time slot synchronization. FDD is particularly advantageous in scenarios with consistent and balanced UL and DL traffic, providing stable and predictable performance. Its inherent ability to support full-duplex operation makes it well-suited for traditional cellular deployments where coverage and capacity are balanced. The primary drawback of FDD lies in its inflexibility to adapt to asymmetric traffic patterns, which are increasingly common in modern applications where downlink demand often exceeds uplink requirements. Additionally, FDD requires paired spectrum, meaning that twice the amount of spectrum is needed compared to TDD for equivalent UL and DL capacities. This can be a limiting factor in spectrum-constrained environments.

In 5G NR, both TDD and FDD are supported for diverse deployment scenarios. TDD is favored in high-frequency bands, such as millimeter wave (mmWave) bands, where the benefits of dynamic UL/DL allocation and advanced antenna technologies are most pronounced. Conversely, FDD remains prevalent in sub-6 GHz bands, where its stability and efficiency in handling balanced traffic make it a reliable choice for wide-area coverage and legacy network integration.

In the context of URLLC, achieving the stringent requirements of low latency and high reliability necessitates advanced configurations of duplexing methods within the RAN. While

FDD offers stable and simultaneous UL and DL communication using separate frequency bands, it lacks the flexibility needed for rapid UL/DL switching. Conversely, Time Division Duplex (TDD) utilizes a single frequency band, dynamically dividing time slots between UL and DL, making it particularly advantageous for latency-sensitive applications such as URLLC. The Fig. 2.8 illustrates Time Division Duplexing (TDD) periodicity configurations used in 5G NR, highlighting the allocation of Downlink (D), Uplink (U), and Special (S) slots for different periodicities (5 ms, 2.5 ms + 2.5 ms, 2.5 ms, and 2 ms). The 5 ms periodicity, the default configuration, provides a stable UL/DL pattern but introduces higher latency, making it suitable for enhanced Mobile Broadband (eMBB). The 2.5 ms and 2 ms configurations reduce periodicity, allowing for faster UL/DL switching and lower latency, which are essential for Ultra-Reliable URLLC. Shorter periodicities improve response times and reduce Round-Trip Time (RTT) but may slightly impact throughput due to more frequent Special (S) slots for guard intervals. This flexibility in periodicity demonstrates how TDD adapts to diverse latency and throughput requirements, making it a cornerstone for real-time 5G applications.

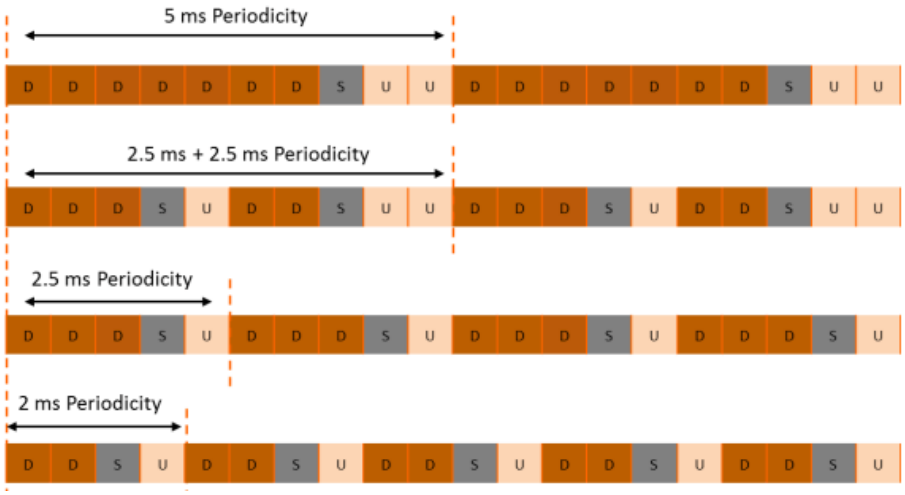
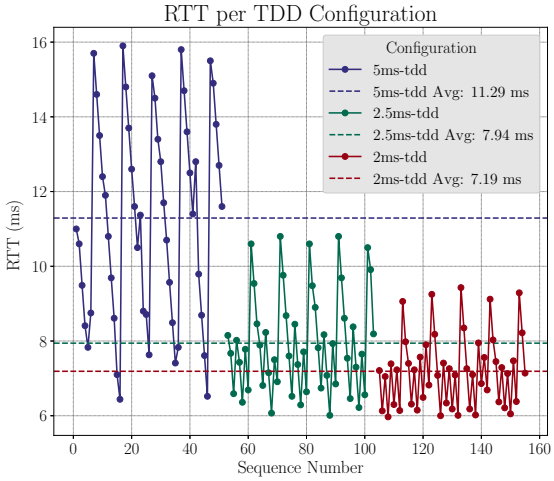


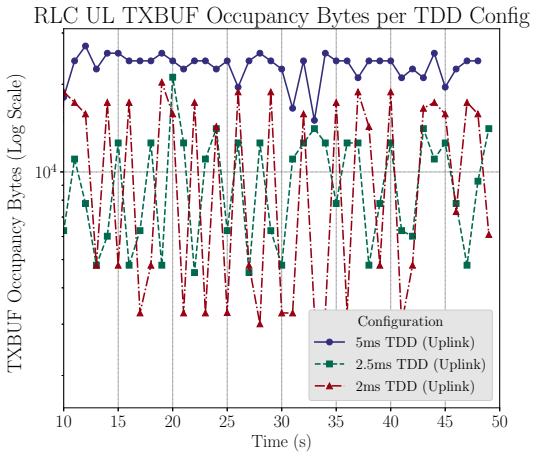
Figure 2.8: TDD periodicities with different configurations.

As highlighted in our study, optimizing TDD configurations plays a pivotal role in reducing RAN latency. By shortening the TDD cycle duration—configuring 2 ms cycles instead of the default 5 ms in OpenAirInterface—we achieved more frequent uplink/downlink switching, facilitating faster response times and lower buffer occupancy in the RLC layer. Fig. 2.9 combines our two key measurements: the Round-Trip Time (RTT) for each TDD periodicity

and the corresponding RLC buffer occupancy in bytes. The 2 ms configuration yields the lowest RTT, making it ideal for URLLC, and also minimizes RLC buffer usage, confirming more efficient data handling under rapid TDD switching. Further improvements were obtained by fine-tuning parameters such as slot-ahead, setting the UL max frame inactivity to zero, and employing a low-latency UPF via DPDK/eBPF.



(α′) Measured RTT for different TDD periodicities.



(β′) RLC buffer occupancy under each TDD cycle.

Figure 2.9: Impact of TDD cycle duration on latency (RTT) and RLC buffer occupancy in an OpenAirInterface 5G testbed.

2.1.6 Software-Defined RAN

Introduction to SDN

Software-Defined Networking (SDN) is a transformative networking paradigm designed to centralize control over network operations, shifting decision-making processes away from individual network devices to a central controller with a global network view. In traditional networks, routers and switches independently make forwarding and routing decisions based on their local configurations. In contrast, SDN decouples these responsibilities by separating the Control Plane (CP) from the User Plane (UP). This separation introduces network programmability, enabling more flexible and dynamic network management [23, 24]. The SDN architecture consists of three key layers: the User Plane (Data Plane), the Control Plane, and the Application Layer. At the lowest layer, the User Plane comprises network devices, such

as SDN-enabled switches, responsible solely for forwarding packets based on rules set by the Control Plane. These switches do not make autonomous routing decisions. If an incoming packet does not match any pre-defined rules in the device's flow table, it is escalated to the central controller via an out-of-band communication channel.

The SDN architecture is illustrated in Fig. 2.10. Above the UP resides the CP, which houses the SDN controller. This centralized controller acts as the brain of the network, responsible for routing, traffic engineering, and policy enforcement. Upon receiving a packet that a switch cannot handle, the controller evaluates the packet using its global network state, determines the optimal path or action, and then instructs the switch on how to forward the packet. Communication between the controller and the User Plane happens through a standardized southbound interface, with OpenFlow emerging as the widely accepted protocol for this purpose. OpenFlow abstracts network devices through flow tables, where each flow entry specifies matching criteria (e.g., IP addresses, VLAN tags) and corresponding actions (e.g., forward to a port, drop, or modify).

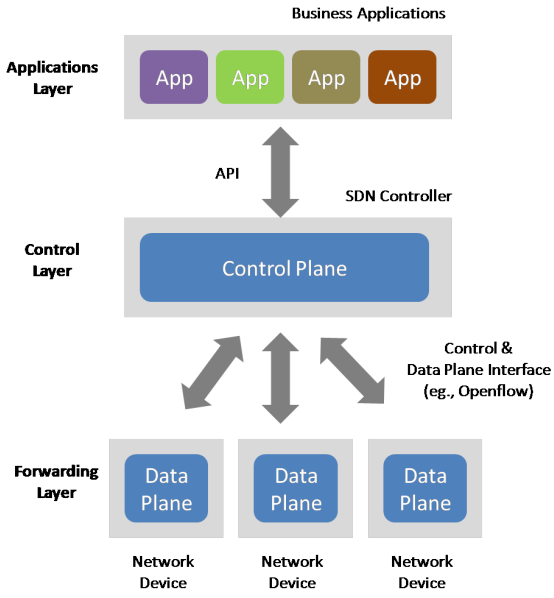


Figure 2.10: SDN Architecture

The Application Layer introduces programmability to the network at the highest level via northbound interfaces. These interfaces allow applications to interact with the SDN controller, facilitating the creation and enforcement of network policies such as load balancing, security rules, or QoS optimizations. The northbound interface abstracts the underlying network complexity, allowing developers to design applications without concern about

hardware-specific configurations. Effectively, the SDN controller functions as a Network Operating System (OS), offering a platform for developing and deploying network applications similarly to traditional OS platforms supporting computer software applications. By decoupling the CP and the UP, SDN offers significant advantages, including centralized visibility, dynamic policy enforcement, simplified network management, and reduced operational costs.

O-RAN

The Open Radio Access Network (O-RAN) represents a paradigm shift in wireless network architectures by introducing openness, flexibility, and programmability into the traditionally closed and vendor-specific RAN ecosystem. By adopting the SDN logic, the fundamental goal of O-RAN is to enable interoperability across multi-vendor components, reduce costs, and foster innovation by disaggregating RAN functions and introducing standardized interfaces [25]. At its core, O-RAN is designed around a service-oriented architecture, focusing on intelligent control, network automation, and AI-driven optimizations.

The architecture is primarily driven by the RAN Intelligent Controller (RIC), which is divided into two functional domains: the Near-Real-Time RIC (Near-RT RIC) and the Non-Real-Time RIC (Non-RT RIC). The Near-RT RIC, as illustrated in Fig. 2.11, executes time-sensitive control tasks, such as traffic steering, mobility management, and interference mitigation, with a latency range of 10ms to 1s. This controller hosts cloud-native microservice-based applications, referred to as xApps, which enhance RAN spectrum efficiency by dynamically adjusting key network parameters, including transmission power and scheduling policies. Meanwhile, the Non-RT RIC, operates at a higher abstraction layer, performing tasks such as policy enforcement, performance analytics, and AI/ML model training. The communication between Non-RT and Near-RT RIC is standardized through the A1 interface.

A critical component enabling communication between the Near-RT RIC and other RAN components is the E2 interface, which serves as a standardized interface connecting the RIC to E2 nodes, including the DU, CU-CP, CU-UP, and even eNB nodes in 4G networks. The E2 interface supports both control procedures and data collection mechanisms across these nodes. It allows the RIC to dynamically manage resource allocation, traffic engineering, and load balancing across cells, slices, QoS classes, or even specific UEs. Furthermore, as it shown from Fig. 2.12 the E2 Application Protocol (E2AP) operates on top of SCTP (Stream Control Transmission Protocol) over IP, ensuring reliable message delivery between the RIC and

E2 nodes. These E2AP messages can embed E2 Service Models (E2SMs), which implement functionalities such as RAN metric collection and control command enforcement, providing granular visibility and dynamic control over network resources. The E2 interface facilitates four primary services provided by the Near-RT RIC: REPORT, CONTROL, INSERT, and POLICY. The REPORT service enables periodic or event-triggered metric reporting from the RAN nodes to the Near-RT RIC, offering real-time visibility into network performance. The CONTROL service dynamically adjusts network parameters to address real-time demands, while the INSERT service allows on-demand configuration changes to optimize performance. Lastly, the POLICY service enforces high-level policies across network slices, QoS classes, and user-specific configurations. These services are combined in Service Models (E2SMs) to simplify and standardize their implementation across different vendor solutions.

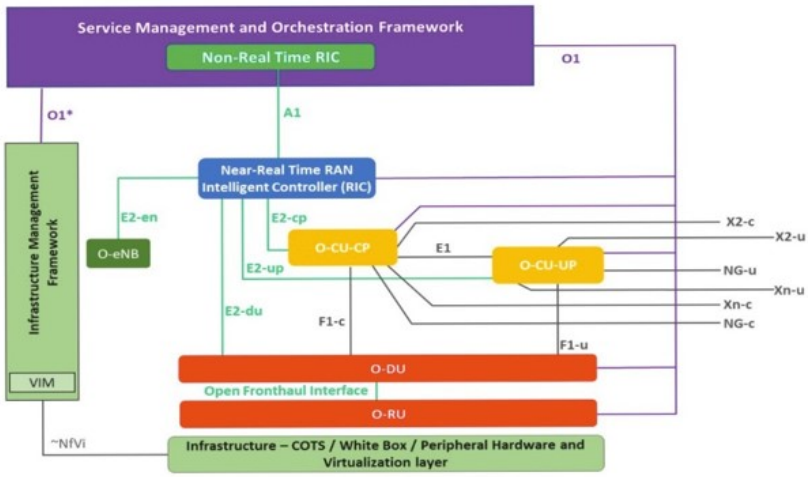


Figure 2.11: The O-RAN Architecture.

Moving to the broader O-RAN architecture, the system is built upon modular components, including the Radio Unit (O-RU), Distributed Unit (O-DU), and Centralized Units (O-CU-CP and O-CU-UP), all connected via the Open Fronthaul Interface. The O-RU handles radio frequency operations, while the O-DU and O-CU perform data processing and control tasks. Communication between these units and the RICs occurs through standardized interfaces, such as E1, F1, and E2. The E2 interface serves as the link between Near-RT RIC and these units, enabling both real-time control and periodic metric reporting. Additionally, the Infrastructure Management Framework, provides virtualization capabilities through Virtual Infrastructure Manager (VIM), allowing dynamic resource allocation across hardware and software components.

The O-RAN architecture emphasizes cloud-native deployments, where network functions are implemented as containerized microservices orchestrated by platforms like Kubernetes. This approach not only ensures scalability and resilience but also simplifies updates and maintenance across distributed network deployments. Moreover, the separation of functionalities across Near-RT RIC, Non-RT RIC, and modular hardware components facilitates a multi-vendor environment, resulting in reducing costs.

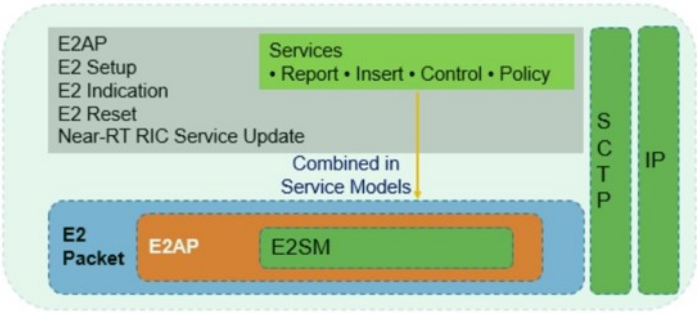


Figure 2.12: E2 Packet Structure.

2.1.7 Key Core Network Functions

In the evolution toward 6G networks, there is a consistent emphasis on embedding intelligence into core network operations. Achieving this requires a comprehensive understanding of the network's current state, including dynamic user demand patterns, precise subscriber locations, and granular network performance metrics. A set of pivotal core network functions collectively contribute to this knowledge layer, enabling data-driven and context-aware decision-making across the network. We focus our study on the following key core network functions:

User Plane Function (UPF)

As discussed in subsection 2.1.1, the UPF plays a pivotal role in managing user data traffic, enforcing policies, and overseeing data forwarding. By continuously monitoring user plane activity, the UPF provides valuable real-time insights into traffic distribution, network load conditions, and user behavior patterns—essential factors for intelligent resource allocation and congestion management. The N3 interface, in particular, can be leveraged for granular flow and packet analysis, offering data that can be fed directly into machine learning models for traffic classification and user demand prediction.

Network Data Analytics Function (NWDAF)

The NWDAF serves as a critical component in the 5G core network architecture, designed to address the increasing complexity of managing dynamic network environments. Positioned within the Service-Based Architecture (SBA) framework, NWDAF functions as an intelligent data analytics engine, collecting, processing, and analyzing vast volumes of network data to enable informed decision-making and efficient network optimization.

At the core of NWDAF lies a modular architecture composed of key functional blocks as illustrated in Fig. 2.13, including the Data Collection Module, Data Storage Layer, Analytics Engine, Model Management, and Exposure Interface. The Data Collection Module interacts with various network functions, AMF, SMF, and Policy PCF, to gather diverse datasets. These datasets include real-time user behavior metrics, traffic flow information, and quality-of-service (QoS) parameters. Once collected, the data is fed into the Data Storage Layer, which serves as a robust repository capable of handling both structured and unstructured data formats. The Analytics Engine, which forms the brain of NWDAF, processes the collected data using advanced statistical analysis and machine learning algorithms. This engine is responsible for identifying patterns, detecting anomalies, and forecasting network behavior, such as predicting congestion hotspots or estimating future resource demands. Machine learning models deployed within NWDAF are continuously trained, validated, and updated by the Model Management component. Data flow within NWDAF follows a well-defined sequence. Initially, the Data Collection Module ingests data from subscribed sources across the network, including metrics from RAN and UE. This data undergoes pre-processing to filter out noise and ensure consistency before being fed into the Analytics Engine. The engine then processes the information, generates predictions, and identifies actionable insights, which are disseminated through the Exposure Interface. Network functions such as the PCF can subsequently use these insights to refine policy control, while the SMF can dynamically allocate resources to address predicted congestion or service degradation.

Location Management Function (LMF)

The LMF is a critical component within the 5GC, primarily responsible for determining and managing the geographical positions of UE. Operating in conjunction with AMF

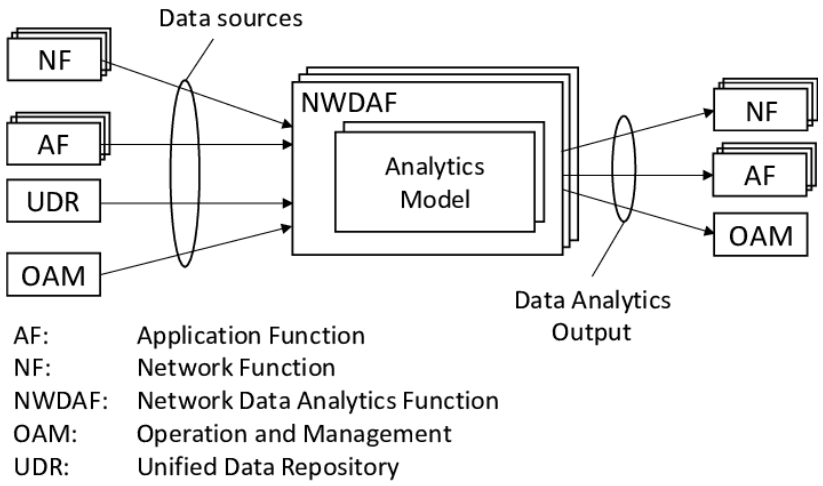


Figure 2.13: NWDAF Architecture in the 5G System.

and the RAN, the LMF facilitates precise positioning services essential for various applications, including emergency services, location-based offerings, and network optimization strategies [26, 27]. A key feature of the LMF is its utilization of multi-cell RTT measurements. This technique involves calculating the time taken for signals to travel from the UE to multiple base stations and back. By analyzing these time measurements from several cells, the LMF can accurately triangulate the UE's position, even in challenging environments where traditional single-cell measurements might be inadequate. Specifically, the multi-cell RTT introduced in 3GPP Release 16 leveraging sounding reference signals (SRS) sent from the UE and Positioning Reference Signals (PRS) received from multiple base stations (BS) as illustrated in Fig. 2.14

Each cell operates independently of the UE's position, enabling high scalability and efficient resource allocation. Multi-Cell RTT also facilitates seamless handovers and dynamic load balancing by predicting UE movement, ensuring optimal connectivity. Additionally, it supports service migration in Mobile Edge Computing (MEC) by dynamically reallocating tasks to the nearest edge servers, reducing latency and enhancing responsiveness. [28].

Beyond positioning, the LMF's capabilities in multi-cell RTT measurements significantly contribute to load balancing during handovers and service migrations. In scenarios where a UE moves through areas served by multiple cells, the LMF assesses real-time location data to predict the UE's trajectory and service requirements. This proximity reduces latency and improves service responsiveness, which is vital for applications requiring real-time data processing.

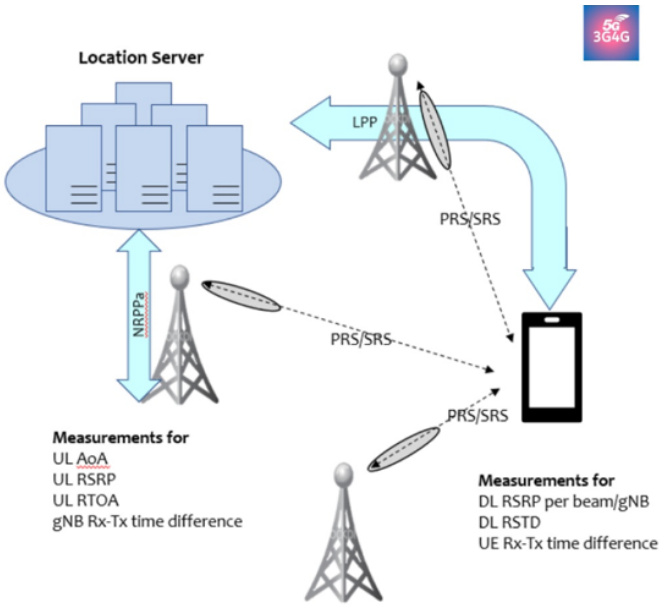


Figure 2.14: Multi-Cell RTT reporting in LMF.

2.2 Multiple Access Edge Computing

2.2.1 Introduction

Multiple Access Edge Computing (MEC) is a transformative paradigm in modern network architectures that aims to bring computational resources, storage, and network services closer to end-users and IoT devices. Unlike traditional cloud computing architectures, where data and applications are processed in centralized data centers, MEC decentralizes these services, enabling real-time data processing at the edge of the network. This proximity reduces latency, enhances bandwidth efficiency, and ensures localized data processing, which is critical for latency-sensitive applications such as autonomous vehicles, augmented reality (AR), virtual reality (VR), and industrial automation [2, 29].

MEC was originally standardized by the European Telecommunications Standards Institute (ETSI) and aimed at mobile networks. Today, it has evolved to support not only mobile networks but also other access technologies such as Wi-Fi, fixed broadband, and satellite communications. MEC nodes are typically deployed within base stations, access points, or regional data centers, allowing them to process and serve user requests more efficiently. The architecture leverages both virtualization technologies and cloud-native principles, enabling

dynamic scalability and resource allocation based on real-time demand.

In essence, MEC bridges the gap between cloud computing and end-users by distributing computational power and storage closer to data sources, thereby addressing key challenges such as network congestion, limited bandwidth, and latency constraints.

2.2.2 Cloud vs Edge

While both cloud computing and edge computing share the goal of delivering scalable computational resources, they differ fundamentally in terms of geographical proximity, latency, and resource management. Cloud computing relies on large, centralized data centers that provide vast computational power and storage capabilities. These data centers are typically located far from end-users, introducing network latency and increased data transmission costs for applications requiring real-time processing [30]. This architecture excels at handling large-scale data analytics, long-term storage, and computationally intensive tasks that are not latency-sensitive.

In contrast, edge computing shifts computational tasks closer to end-users, often at the network edge or near access points such as base stations, access routers, or localized edge servers. This architectural difference drastically reduces latency, making edge computing ideal for latency-sensitive applications such as URLLC, industrial IoT, and smart city infrastructures. By processing data closer to the source, edge computing minimizes the delays associated with data traversal across long network paths and reduces congestion on core networks [31].

For instance, in a cloud-based system, data generated by autonomous vehicles must travel long distances to centralized cloud servers for processing and analysis. This introduces delays that can be detrimental in time-critical scenarios, such as collision avoidance. In an edge-based system, the same data can be processed locally at a MEC node within milliseconds, enabling faster decision-making and ensuring safer vehicle operations. Moreover, data privacy and security are often better managed at the edge, as sensitive data can be processed and analyzed locally rather than being transmitted over potentially insecure long-distance connections to centralized cloud data centers.

Despite their differences, cloud computing and edge computing are not mutually exclusive technologies. Instead, they can work synergistically in a hybrid cloud-edge architecture to deliver optimal performance, efficiency, and scalability. In this hybrid model, latency-

sensitive tasks are processed at the edge, while resource-intensive tasks—such as large-scale data analytics, AI model training, or archival storage—are offloaded to centralized cloud infrastructures.

2.2.3 Placing MEC in Telecom Networks

The deployment of MEC has its origins in the architectures of LTE/4G networks, where ETSI proposed various strategic placements for MEC hosts. These placements aimed to balance latency, network efficiency, and resource accessibility based on their physical location within the network architecture [32].

One of the initial deployment models was MEC over the SGi interface. In this architecture, the MEC host is positioned on the network's backhaul, specifically before the Serving Gateway and Packet Gateway on the SGi interface. As shown in Fig. 2.15, placing MEC closer to the core network offers more accessible services to network users but introduces a significant drawback in terms of latency. The RTT increases due to the physical distance between the UE and the core network infrastructure. As a result, this setup resembles traditional cloud computing models, where latency-sensitive applications may not see substantial performance improvements. The UL flow in this setup follows the sequence: UE → eNodeB → CORE → MEC.

To address the limitations of SGi placement, MEC over the S1 interface was introduced as an alternative deployment model. In this architecture, the MEC host is positioned between the eNodeB and the centralized core site, as shown in Fig. 2.16. By being closer to the Fronthaul network, MEC reduces RTT latency, enhancing the responsiveness of time-sensitive applications. With this placement, packet flows follow the sequence: UE → eNodeB → MEC → CORE. The proximity of the MEC to the user equipment significantly reduces latency and improves the overall QoS for real-time applications.

In the evolution towards 5G and Beyond-5G networks, MEC placements have adapted to more disaggregated architectures, leveraging the separation of CUs and DUs. A prominent approach involves deploying MEC hosts directly adjacent to DUs on the Fronthaul network [33]. In this architecture, the UL path is configured as: UE → DU → MEC → CU, as

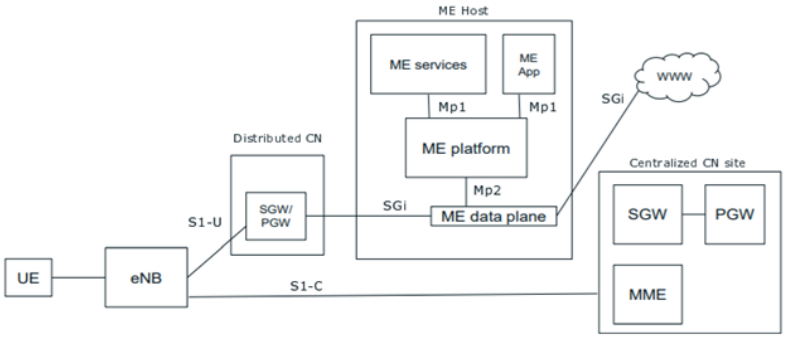


Figure 2.15: MEC on the SGi interface.

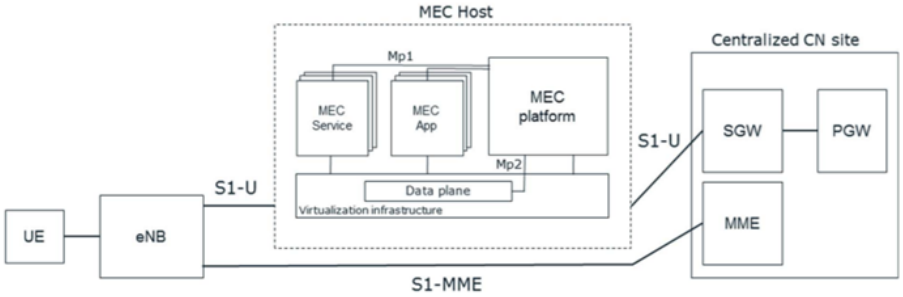


Figure 2.16: MEC on the S1 interface.

illustrated in Fig. 2.17. This configuration enables ultra-low latency communications, particularly beneficial for services requiring near-instantaneous processing, such as vehicle-to-everything (V2X) and augmented reality (AR) applications. Communication in this setup relies heavily on the F1 Application Protocol, which standardizes interactions between CUs and DUs. The MEC Agent, integrated with the MEC host, plays a critical role in facilitating communication between DUs and MEC services. Specifically, the MEC Agent manages the Radio Network Temporary Identifiers (RNTIs) for the UEs, ensuring that requests and responses are accurately mapped between the DUs and the MEC services. When a DU has data packets to transmit to the MEC, it generates a MEC Data Request message. The MEC Agent intercepts this message, processes it, and routes the payload (user data packets) to the appropriate MEC service. Similarly, when data is sent from a MEC service back to a UE, the MEC Agent generates a MEC Data Indication message to ensure delivery to the appropriate DU.

The UPF can also act as a MEC server, offering enhanced capabilities for edge computing workloads. The UPF serves as a gateway that handles user plane traffic, enabling traffic

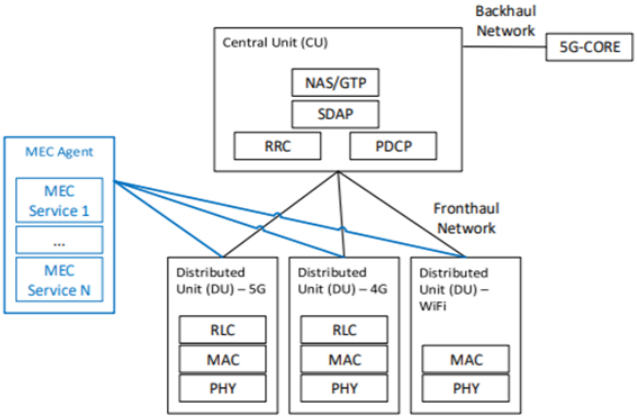


Figure 2.17: Placing MEC next to DUs.

steering, routing, and filtering functionalities. Placing MEC servers adjacent to or integrated with UPFs creates a highly efficient architecture where user data packets can be processed locally at the edge of the network, reducing backhaul traffic and minimizing latency. Additionally, MEC can also be colocated with the CU-UP to further optimize traffic management and processing. The CU-UP is responsible for managing user-plane functionalities such as packet forwarding and QoS enforcement. By integrating MEC services directly with CU-UP nodes, user data can bypass unnecessary network hops, allowing ultra-fast processing at the edge. Specifically, in centralized deployments, UPFs support high-capacity applications (e.g., eMBB) but suffer from higher latency. In contrast, localized deployments colonize MEC, UPF, and CU-UP near DUs, ensuring ultra-low latency for services such as URLLC and industrial automation, as illustrated in Fig. 2.18.

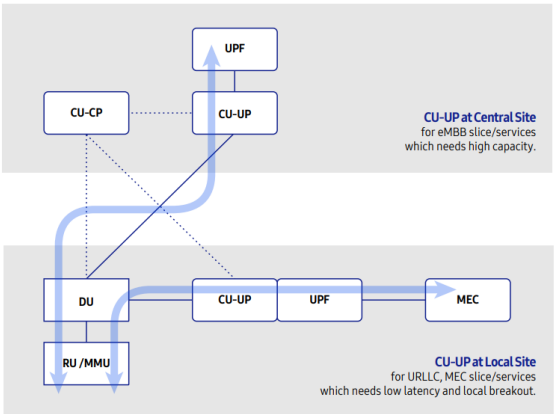


Figure 2.18: MEC Deployments in beyond 5G Networks.

2.2.4 MEC Type Deployment - Virtualization Technologies

The deployment of a MEC host relies heavily on virtualization technologies to ensure scalability, flexibility, and efficient resource utilization across distributed edge nodes. Virtualization enables MEC to host diverse services and applications dynamically while optimizing hardware resources. This subsection explores the key virtualization technologies employed in MEC deployments, including Virtual Machines (VMs), Containers, and supporting frameworks such as Network Function Virtualization (NFV) and Service Orchestration. In the following, we list the main technologies:

Virtual Machines: VMs are one of the earliest virtualization technologies used in MEC deployments. A VM emulates a complete hardware environment, allowing multiple operating systems and applications to run on a single physical server. VMs are managed by hypervisors such as KVM (Kernel-based Virtual Machine) and VMware ESXi, which enable hardware-level isolation and resource allocation. In the MEC context, VMs are often used for applications requiring strong isolation, such as security services, mission-critical workloads, or legacy applications that depend on specific OS configurations. However, VMs have relatively higher overhead in terms of memory and CPU usage, making them less suitable for resource-constrained edge environments.

Containers: As observed in Fig. 2.19, in contrast to VMs, containers offer lightweight virtualization by sharing the host operating system's kernel while isolating applications and their dependencies. Containers are characterized by low resource overhead, faster startup times, and high portability, making them ideal for microservice-based MEC architectures. Each MEC application can run as an independent containerized service, enabling rapid scaling, efficient resource allocation, and simplified deployment across distributed edge nodes. For example, a video analytics application deployed in MEC can scale container instances dynamically based on demand, ensuring uninterrupted performance during traffic surges.

Network Function Virtualization: NFV plays a crucial role in MEC by decoupling network functions, such as firewalls, load balancers, and packet inspection tools, from dedicated hardware appliances. These network functions are deployed as Virtual Network Functions

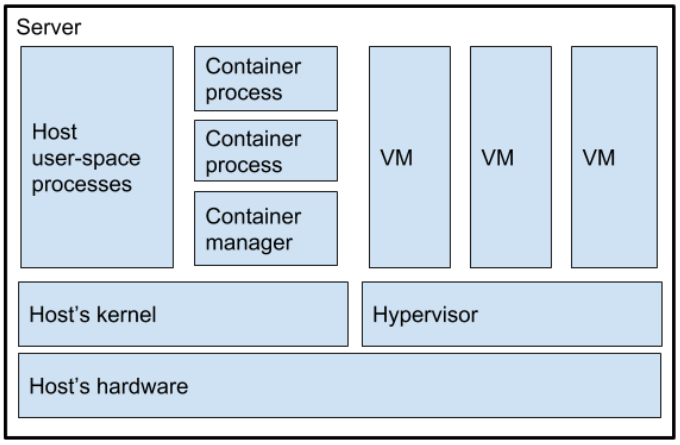


Figure 2.19: VMs vs Containers.

(VNFs) on MEC servers. NFV enables dynamic deployment and scaling of VNFs across edge nodes, ensuring that network resources are allocated efficiently based on traffic patterns and service demands. In MEC deployments, NFV allows service providers to create virtualized instances of edge services on demand, improving flexibility and resource utilization.

Service Orchestration: To manage virtualized resources and services in MEC deployments, orchestration frameworks such as OpenStack and Kubernetes are widely used. These platforms provide end-to-end automation for deploying, scaling, and managing MEC applications. Kubernetes (will be discussed further in the 2.4 section), in particular, has become the de facto standard for managing containerized applications in MEC environments due to its robust orchestration capabilities, self-healing mechanisms, and seamless scaling features.

2.2.5 Edge Service Live Migration

Edge Service Live Migration is a critical capability in MEC that enables the seamless transfer of running services, applications, or workloads from one edge node to another without disrupting their operation. This functionality is essential in dynamic network environments, where factors such as user mobility, resource availability, and network congestion require flexible and real-time resource reallocation. Live migration ensures service continuity, low latency, and optimized resource utilization. In edge computing environments, end-users often move across geographical regions. If a service is anchored to a single MEC node, user mobility can result in increased latency and degraded service performance as the physical distance between the user and the service-hosting MEC node increases. Live migration ad-

dresses this challenge by dynamically transferring workloads to the nearest MEC node, maintaining optimal performance and uninterrupted service delivery. Additionally, live migration is beneficial in scenarios where an MEC node becomes overloaded due to high computational demand or faces resource constraints. By offloading certain services to less congested MEC nodes, network operators can balance workloads effectively, preventing performance bottlenecks.

During live migration, the ideal scenario is to maintain critical states, including kernel state, active TCP/IP connections, application state, and sockets. However, maintaining these states can face significant challenges, such as long downtime caused by the large number of memory pages that must be copied between the source and target MEC nodes. Furthermore, network connectivity between the source and target nodes plays a crucial role in determining migration success. These constraints have led to two main categories of migration: stateful migration and stateless migration. In stateful migration, the application and network connection states are preserved, ensuring a seamless transition with minimal impact on service availability. Conversely, in stateless migration, network connections are lost, but memory pages and disk contents are successfully transferred, allowing the application to resume on the target node.

Service live migration in MEC relies on VMs and containers, with support from NFV and SDN. VM-based migration, managed by hypervisors like KVM and VMware ESXi, offers strong isolation and reliable resource encapsulation. However, it remains resource-intensive and may introduce latency during migration, making it less suitable for latency-sensitive services. In contrast, container-based migration, facilitated by platforms such as Docker and Kubernetes, provides a lightweight alternative with faster recovery times and minimal overhead. Tools like CRIU (Checkpoint/Restore In Userspace) enable the state of running containers, including open files, memory state, and network connections, to be saved and restored during migration. This approach significantly reduces downtime and ensures smoother transitions.

Having a VM Live Migration as a reference, the live migration process typically involves three main phases:

1. **Pre-Migration Phase:** The MEC system analyzes parameters such as *service state*, *resource availability*, and *network conditions* to determine whether live migration is necessary. Migration policies, such as maintaining latency thresholds or resource load balancing, guide this decision.

2. **Migration Phase:** During this phase, the active state of the service is transferred from the *source MEC node* to the *target MEC node*. Depending on the virtualization technology migration can follow different approaches:

    - **Pre-copy Migration:** In this method, memory pages from the source VM are copied iteratively to the target node while the VM continues to run on the source node. During the initial phase, modified memory pages (dirty pages) are tracked and transferred incrementally to the target node. Once the number of dirty pages stabilizes and becomes smaller than the number of pages transferred per iteration, the VM is paused, and the remaining memory pages are transmitted to the target node. The VM is then resumed on the target node with minimal downtime. Pre-copy migration has the advantage of shorter perceived downtime but often results in a higher total data transfer volume.
    - **Post-copy Migration:** In contrast, post-copy migration begins by pausing the VM on the source node and transferring only the minimal processor state to the target node. Once the VM resumes execution on the target node, the remaining memory pages are fetched on demand from the source node over the network. While post-copy migration reduces the total amount of data transferred, it is more prone to performance degradation if network instability or connectivity issues arise during the migration process

3. **Post-Migration Phase:** After the migration, the target node takes over the service, and final consistency checks are performed to ensure that all states and dependencies have been accurately transferred. The source node then releases its resources.

As shown in Fig. 2.20, pre-copy migration typically results in a larger data transfer volume compared to post-copy migration, but it benefits from shorter service downtime due to adaptive algorithms for managing dirty pages [34]. On the other hand, post-copy migration minimizes data transfer but risks service interruption if critical memory pages are delayed during retrieval.

While live migration offers significant benefits, it also introduces several challenges. Latency sensitivity is critical, especially for URLLC applications, where even minimal delays

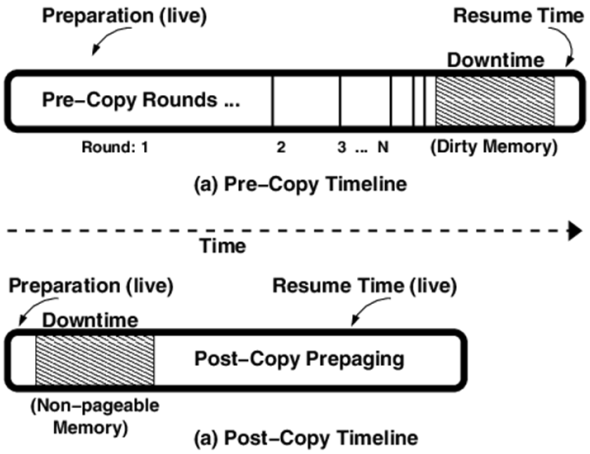


Figure 2.20: Pre-Copy vs Post-Copy Migration.

can disrupt performance. Maintaining data consistency between the source and target nodes is equally important, particularly for stateful applications handling transactional data. Additionally, resource constraints at edge nodes, with limited computational power and storage, make it difficult to allocate resources without disrupting other services. Finally, network bandwidth is crucial, as insufficient capacity can prolong migration times and increase service downtime.

2.3 Artificial Intelligence and Machine Learning Introduction

AI and ML techniques have increasingly gained significance in addressing complex and dynamic problems encountered within telecommunication networks. By providing the capability to automatically analyze data, identify intricate patterns, and make intelligent decisions, these techniques are fundamental for optimizing telecommunication network performance, reliability, and adaptability. In this section, we present a thorough review of the core principles and architectures employed within AI/ML, focusing particularly on the methodologies applied throughout this thesis, including classical machine learning, deep learning with emphasis on time-series prediction, and reinforcement learning.

2.3.1 Machine Learning (ML)

ML algorithms enable systems to learn from data patterns, improving predictions through iterative training. In supervised learning, algorithms minimize a loss function representing prediction error. A common loss function, Mean Squared Error (MSE), is defined as:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f_\theta(x_i))^2$$

where N is the number of training samples, y_i the true label, $f_\theta(x_i)$ the predicted output of the model parameterized by θ, and $(y_i - f_\theta(x_i))^2$ measures the squared difference between predictions and actual labels.

Support Vector Machines (SVM)

Support Vector Machines are powerful classification algorithms widely used for tasks such as intrusion detection and anomaly identification in telecommunications. The fundamental idea of an SVM is to identify the optimal hyperplane that best separates data points belonging to different classes. This hyperplane maximizes the margin between data points from each class, providing robust generalization capability.

Formally, consider a dataset composed of data points (x_i, y_i), where $x_i \in \mathbb{R}^d$ is the feature vector and $y_i \in \{-1, +1\}$ is the corresponding class label. An SVM identifies the separating hyperplane defined by:

$$w \cdot x + b = 0 \tag{2.2}$$

where:

- w is the normal vector to the hyperplane (decision boundary),
- x is the input feature vector,
- b is the bias term (intercept).

The SVM optimization problem seeks to maximize the margin, which translates into minimizing the magnitude of the vector w. The formal optimization problem can be written as:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \tag{2.3}$$

subject to the constraints:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, ..., N \tag{2.4}$$

where:

- $\|w\|$ is the Euclidean norm of the vector w, reflecting the complexity of the model.
- C is a regularization parameter controlling the trade-off between maximizing the margin and minimizing the classification errors. A larger C imposes a stricter penalty on misclassification.
- ξ_i are slack variables introduced to handle misclassifications and data points that fall within the margin boundaries, allowing the model to cope with noise or non-linear separations.

This optimization problem is solved using Quadratic Programming (QP) techniques, resulting in an optimal hyperplane with strong classification performance even in high-dimensional feature spaces. Furthermore, kernel methods, such as the Gaussian Radial Basis Function (RBF), can be employed to map data into higher-dimensional spaces to enable nonlinear classification:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \tag{2.5}$$

where the parameter γ defines the spread of the kernel function.

Random Forest (RF)

Random Forests are ensemble methods that consist of multiple decision trees trained on various subsets of the data. RFs are especially useful due to their ability to handle large datasets with high-dimensional feature spaces, providing robust predictions while mitigating overfitting.

Formally, an RF model aggregates predictions from individual decision trees as follows:

$$f_{RF}(x) = \frac{1}{T} \sum_{t=1}^{T} f_{\text{tree},t}(x) \tag{2.6}$$

where:

- T represents the total number of decision trees.
- $f_{\text{tree},t}(x)$ is the prediction made by the t^{th} decision tree for the input vector x.

Each tree is constructed using a random subset of features and a bootstrapped sample of the data points, thus encouraging diversity among individual trees and improving generalization. Important hyperparameters include the number of trees (T), maximum depth of trees, and the number of features randomly selected for splits.

In this thesis, ML techniques such as SVM, Random Forest, and Autoencoders form a critical foundation for predictive and anomaly detection tasks within network infrastructures, analyzed further in Chapter 6. Specifically, SVM and RF models are extensively utilized for intrusion detection and security enhancement. By systematically selecting parameters such as C and γ (for SVM), the number and depth of trees (for RF) we utilize these algorithms to achieve optimized performance in diverse real-world scenarios.

2.3.2 Deep Learning (DL) and Neural Networks

Deep Learning (DL) leverages hierarchical neural architectures—ranging from simple feed-forward networks to sophisticated recurrent and convolutional models—to learn rich representations from raw data. In the context of 6G networks, these models enable accurate forecasting of traffic loads, real-time adaptation of radio resources, and robust anomaly detection. Below we describe key DL architectures, their mathematical foundations, and how each is applied to specific networking tasks.

Feed-Forward Neural Networks (FNNs)

A Feed-Forward Neural Network (FNN) comprises an input layer, one or more hidden layers, and an output layer, with information flowing in one direction. Mathematically, for an L-layer FNN:

$$h^{(0)} = x, \quad h^{(\ell)} = f\big(W^{(\ell)}\, h^{(\ell-1)} + b^{(\ell)}\big),\ \ell = 1, \ldots, L, \quad \hat{y} = W^{(L+1)}\, h^{(L)} + b^{(L+1)}.$$

Here x is the feature vector (e.g., instantaneous bandwidth usage, CQI values), $W^{(\ell)}, b^{(\ell)}$ are learned parameters, and f is a nonlinear activation. The network is trained to minimize a loss $\mathcal{L}(y, \hat{y})$ via backpropagation:

$$\theta \leftarrow \theta - \eta\, \nabla_\theta \mathcal{L},$$

where $\theta = \{W^{(\ell)}, b^{(\ell)}\}$. FNNs serve as efficient baselines for static tasks in this thesis, such as coarse classification of traffic types or regression of average throughput under moderate variability.

Recurrent Neural Networks (RNNs)

RNNs introduce internal state to model sequences $\{x^{(1)}, \ldots, x^{(T)}\}$. At each time step t:

$$h^{(t)} = f\big(W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h\big), \quad y^{(t)} = g\big(W_{hy}h^{(t)} + b_y\big),$$

where $h^{(t)}$ captures information from all previous inputs. RNNs are thus well suited for time-series forecasting of network metrics—such as short-term traffic spikes—where the immediate past strongly influences predictions. However, standard RNNs struggle with long-range dependencies.

Long Short-Term Memory (LSTM)

LSTM networks overcome RNN limitations via gated memory cells. Each cell computes:

$$\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f), \\
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i), \quad \tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C), \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o), \quad h_t = o_t \odot \tanh(C_t).
\end{aligned}$$

Here, f_t (forget gate) decides what past information to retain, i_t (input gate) regulates new information, and o_t (output gate) controls exposure of memory. LSTMs excel in capturing long-term patterns—essential for anticipating recurrent traffic patterns (e.g. daily peak hours) and supporting proactive slice reconfiguration in Chapter 5.

Gated Recurrent Unit (GRU)

GRUs simplify LSTMs by combining gates:

$$\begin{aligned}
z_t &= \sigma(W_z[h_{t-1}, x_t] + b_z), \quad r_t = \sigma(W_r[h_{t-1}, x_t] + b_r), \\
\tilde{h}_t &= \tanh(W_h[r_t \odot h_{t-1},\, x_t] + b_h), \quad h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t.
\end{aligned}$$

The update gate z_t and reset gate r_t control memory flow, reducing complexity while maintaining performance. GRUs are particularly effective when computational resources are limited (e.g. at edge nodes) but sequence modeling beyond a few time steps remains critical.

Bidirectional RNNs (Bi-RNNs)

Bi-RNNs process sequences in both forward and backward directions:

$$\overrightarrow{h}^{(t)} = \mathrm{RNN}(x^{(t)}, \overrightarrow{h}^{(t-1)}), \quad \overleftarrow{h}^{(t)} = \mathrm{RNN}(x^{(t)}, \overleftarrow{h}^{(t+1)}),$$

and concatenate $h^{(t)} = [\overrightarrow{h}^{(t)}; \overleftarrow{h}^{(t)}]$. This dual context is invaluable when slice decisions depend on both past trends and anticipated near-future conditions.

Convolutional Neural Networks (CNNs)

CNNs capture local patterns via convolutional filters. A 1D convolution over temporal data is:

$$y_{i,k}^{(\ell)} = f\Big(\sum_{c=1}^{C_{\ell-1}} \sum_{m=1}^{M} W_{m,c,k}^{(\ell)}\, x_{i+m-1,c}^{(\ell-1)} + b_k^{(\ell)}\Big).$$

Here, $W^{(\ell)}$ filters detect motifs such as sudden traffic bursts across short windows. In Chapter 5, CNNs serve as front-end feature extractors before feeding into LSTM layers, improving detection of localized traffic anomalies.

Autoencoders

Autoencoders learn compact encodings h of input x and reconstruct $\hat{x}$:

$$h = f_{\mathrm{enc}}(x), \quad \hat{x} = f_{\mathrm{dec}}(h), \quad \mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|_2^2.$$

Variants include:

- *Sparse Autoencoders*, adding a sparsity penalty $\sum_i \mathrm{KL}(\rho \parallel \hat{\rho}_i)$.
- *Denoising Autoencoders*, which reconstruct clean x from corrupted inputs $\tilde{x}$.

In Chapter 6, autoencoders isolate anomalous network states: large reconstruction errors signal deviations from learned normal patterns, triggering security xApp responses.

2.3.3 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning that involves training agents to make decisions by interacting with an environment to achieve specific objectives. The agent learns optimal actions based on rewards received through these interactions, thus

maximizing long-term cumulative rewards. RL has been widely adopted in dynamic and sequential decision-making problems, such as telecommunications, autonomous driving, robotics, and resource management in networks.

Core Concepts of Reinforcement Learning

The Reinforcement Learning framework comprises four fundamental elements: the *agent*, the *environment*, *states*, and *actions* as depicted in Fig. 2.21. At each discrete timestep t, the agent observes the current state $s_t \in S$ of the environment and selects an action $a_t \in A$ according to a policy π. After executing this action, the agent receives a scalar reward $r_t \in \mathbb{R}$ and transitions to a new state s_{t+1}. This process repeats iteratively, generating a sequence of states, actions, and rewards known as a trajectory or episode.

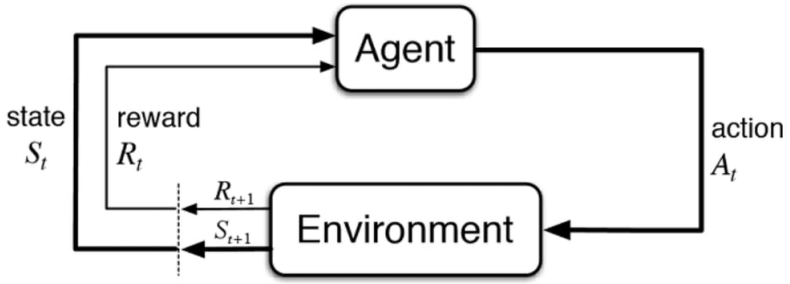


Figure 2.21: Overview of RL.

The primary goal of an RL agent is to discover an optimal policy π^* that maximizes the expected cumulative discounted reward, defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

where $\gamma \in [0, 1)$ is the discount factor, balancing immediate versus future rewards.

Markov Decision Processes (MDP)

Formally, an RL task is typically modeled as a Markov Decision Process (MDP), represented by the tuple (S, A, P, R, γ), where:

- S denotes the finite set of states.

- A denotes the finite set of actions.
- $P(s_{t+1}|s_t, a_t)$ is the state-transition probability, specifying the likelihood of transitioning to state s_{t+1} given the current state s_t and action a_t.
- $R(s_t, a_t)$ is the reward function, providing a scalar feedback after taking action a_t in state s_t.
- γ is the discount factor, determining the relative importance of future rewards.

Value Functions and Bellman Equations

A critical concept in RL is the notion of *value functions*, quantifying the expected return of following a policy π. The state-value function $V_\pi(s)$ for a given policy π is defined as:

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | s_t = s \right]$$

Similarly, the action-value function $Q_\pi(s, a)$ represents the expected return of choosing action a in state s under policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | s_t = s, a_t = a \right]$$

Both value functions satisfy recursive relationships known as the Bellman equations. For the state-value function, the Bellman equation is:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} P(s', r|s, a)[r + \gamma V_\pi(s')]$$

For the optimal action-value function $Q^*(s, a)$, the Bellman optimality equation is defined as:

$$Q^*(s, a) = \sum_{s',r} P(s', r|s, a) \left[r + \gamma \max_{a'} Q^*(s', a') \right]$$

Deep Reinforcement Learning

When dealing with large and complex environments, traditional tabular RL methods become impractical. Deep Reinforcement Learning (DRL) emerges as a powerful approach by integrating deep neural networks as function approximators for value or policy functions. Prominent DRL algorithms include Deep Q-Network (DQN), Deep Deterministic Policy Gradient (DDPG), and Advantage Actor-Critic (A2C).

For instance, DQN leverages neural networks to approximate the optimal action-value function $Q^*(s, a; \theta)$ with network parameters θ. The network is trained by minimizing the loss function:

$$\mathcal{L}(\theta) = \mathbb{E}\left[(y_t - Q(s_t, a_t; \theta))^2\right]$$

where y_t is the target value, computed as:

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)$$

and θ^- denotes parameters of a separate target network periodically updated from θ to stabilize learning.

Within this thesis, DRL is applied to telecommunications scenarios, particularly focusing on Multi-access Edge Computing (MEC) service migration that presented in chapter 4. In MEC contexts, the DRL agent relocates edge services based on users' real-time mobility patterns and network conditions, significantly reducing latency and enhancing service continuity.

2.4 Experimental Tools and Methods

2.4.1 SLICES RI - Testbeds

SLICES Research Infrastructure (SLICES RI) [35] is a cutting-edge, large-scale experimental platform designed to support research and innovation in networking and distributed systems, specifically focusing on 5G, 6G, and beyond. As an advanced testing environment, SLICES RI enables researchers to design, implement, and evaluate new technologies and protocols under real-world conditions while leveraging its distributed architecture for scalability and diversity. The platform fosters collaboration by offering open access to resources for academia, industry, and research institutions. SLICES RI integrates state-of-the-art testbeds, providing support for a wide range of experiments in network virtualization, MEC, network slicing, and cloud-native deployments. The infrastructure spans multiple geographically distributed sites, ensuring a heterogeneous environment for testing different network configurations and deployment strategies.

In our experiments, we utilized two key testbeds that are part of SLICES RI: One Lab - Sorbonne University [36] and the NITOS testbed [37] in Greece. These facilities provided

the necessary infrastructure and flexibility to evaluate and validate our solutions in diverse network environments.

- **NITOS Testbed:** It is hosted by the University of Thessaly in Greece, as a primary facility for development and testing. NITOS is a remotely accessible, 24/7 testbed, designed for advanced experimentation in wired and wireless networks. iT consists of over 100 high-performance nodes, each equipped with Core-i7 processors, GPUs for AI/ML experimentation, and multiple IEEE 802.11 a/b/g/n/ac wireless cards for WiFi research, supporting open-source drivers like ath9/10k. Furthermore, LTE/5G research capabilities are supported through more than 20 Software-Defined Radio (SDR) devices, enabling customizable RAN operations, and six mmWave devices for creating high-throughput wireless links and facilitating beam steering for various topologies. All nodes connect through a tree-structured OpenFlow-enabled network comprising hardware switches, ensuring seamless connectivity across the testbed. The testbed is depicted in Fig. 2.22, while its architectural design is detailed in Fig. 2.23.

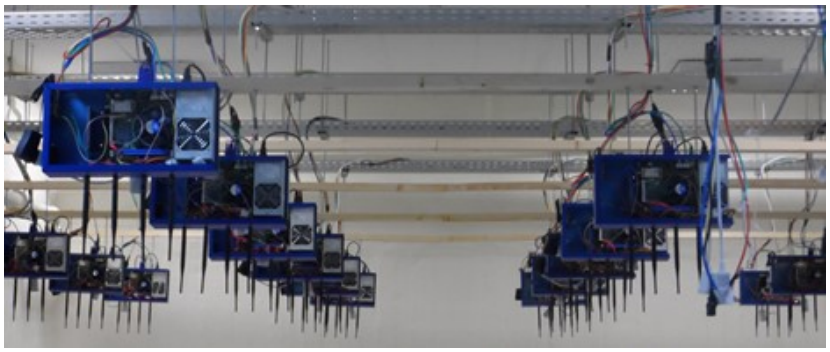


Figure 2.22: Overview of the NITOS testbed.

- **One Lab - Sorbonne University:** This testbed offers advanced resources for conducting experiments on *AI/ML-driven network optimizations* and *multi-access edge computing*. It is equipped with small cells, high-performance computing servers, and P4 switches.

2.4.2 5G Experimentation Tools

To evaluate and implement advanced concepts in 5G networks, this thesis relies on state-of-the-art experimentation tools. These tools facilitate the development, testing, and opti-

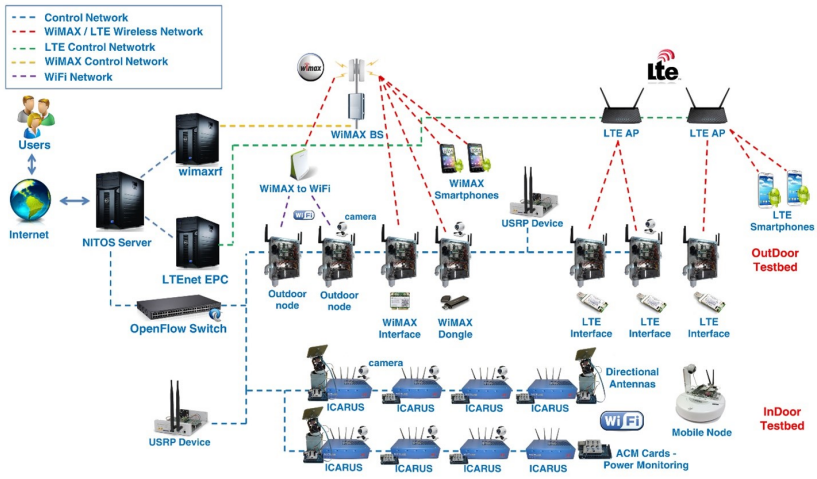


Figure 2.23: NITOS testbed Nodes.

mization of various 5G components, including RAN, core network functionalities, and control frameworks.

OpenAirInterface

OpenAirInterface [38] is an open-source software platform that provides a fully functional implementation of 5G and 4G Radio Access Network (RAN) and Core Network (CN) components. Developed by the OpenAirInterface Software Alliance (OSA), OAI supports research and prototyping in real-time and simulated 5G environments. OAI follows 3GPP standards and functionalities such as RAN slicing, dynamic resource allocation, and user mobility management, making it a versatile tool for testing innovative solutions in 5G networks. OAI's modular architecture allows for seamless integration with other 5G components and supports deployment on Software-Defined Radios (SDRs) or simulated environments. Its compatibility with 3GPP standards ensures researchers can evaluate new functionalities while adhering to industry requirements.

FlexRAN

FlexRAN [39] is an open-source Software-Defined RAN platform designed to bring programmability and flexibility to RANs by decoupling the control plane from the data plane. It introduces a hierarchical architecture composed of two main components: the FlexRAN Service and Control Plane and the FlexRAN Application Plane. The Service and Control Plane features a Real-Time Controller that communicates with underlying RAN runtimes, which

serve as abstraction layers for different RAN modules, such as monolithic 4G eNodeBs or disaggregated 4G and 5G deployments. FlexRAN ensures efficient communication between the RTC and the RAN agent within the runtime environment, enabling a wide range of RAN control actions. These actions, enable functionalities like real-time monitoring, traffic steering, mobility management, and dynamic RAN slicing. FlexRAN also integrates seamlessly with OAI, enhancing its utility for end-to-end 5G experimentation. Its programmability enables researchers to implement and test custom algorithms for a variety of use cases, including traffic steering and resource slicing.

FlexRIC

FlexRIC [40] is a next-generation RIC platform designed for Open RAN (O-RAN) architectures. It provides both Near-RT and Non-RT control functionalities, enabling dynamic optimization of RAN operations. FlexRIC supports the deployment of customized applications (xApps and rApps) for tasks such as traffic prediction, resource allocation, and anomaly detection through customed or standardized SMs. Its adherence to O-RAN Alliance specifications ensures interoperability with diverse network components. Furthermore, FlexRIC's modular design and support for E2 interfaces make it an ideal tool for evaluating O-RAN-based solutions in 5G networks. FlexRIC supports multi-vendor integration, including OAI and srsRAN, allowing it to function in diverse network environments. It extends the capabilities of the O-RAN E2 interface through the introduction of the E42 interface, which includes additional procedures such as E42 Setup Request, E42 Setup Response, E42 Subscription Request, E42 Subscription Delete Request, and E42 RIC Control Request. The Fig. 2.24 illustrates the architecture of the FlexRIC controller system and its interaction within the O-RAN ecosystem. The controller comprises components like xApps, a database, communication libraries, intelligent applications (iApps), and a server library.

2.4.3 Kubernetes Ecosystem

In recent years, microservices—driven by containerization—have become a preferred approach for creating highly scalable and portable applications. In telecommunications and computer networks, Network Function Virtualization (NFV) integrates seamlessly with vir-

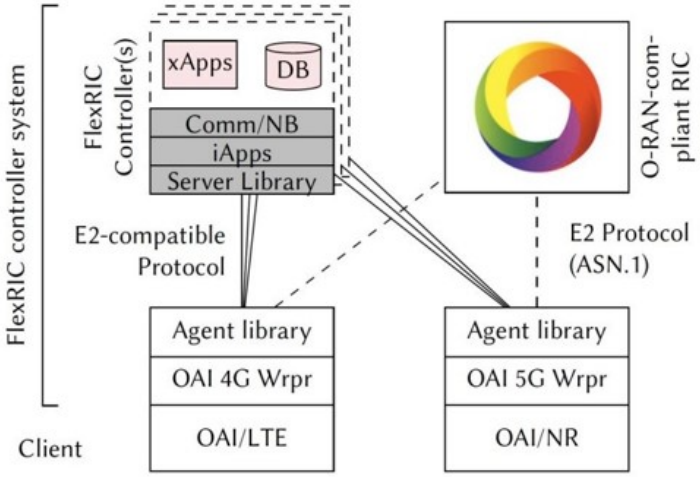


Figure 2.24: FlexRIC Architecture.

tual machines, enabling efficient representation of network functions. The Kubernetes framework, with its robust API, orchestrates both containers and virtual machines through the KubeVirt add-on, which encapsulates existing virtual machines within containers. This orchestration reduces operational costs and complexity for cloud-native network functions, positioning Kubernetes as both a Virtual Infrastructure Manager (VIM) and a Virtual Network Function Manager (VNFM). Leveraging NFV, beyond 5G networks can be fully software-defined, enabling virtualization of all functional components. Consequently, deploying 5G/6G on Kubernetes streamlines resource management, monitoring, scalability, and installation. Therefore, we examine the Kubernetes ecosystem, along with integrated technologies, in support of the objectives of the dissertation.

Introduction to Docker

Docker [41] is a leading containerization platform that facilitates the creation and management of isolated application environments known as containers. Containers encapsulate an application's code along with all its dependencies, ensuring consistency and portability across diverse environments, including data centers, cloud platforms, and personal computers. Docker primarily consists of two fundamental components:

- Docker Containers: These are lightweight, standalone units that package the application code and its necessary dependencies. Containers operate in isolation from the host environment, containing only the essential operating system components, libraries, and services required for the application to function. This isolation allows multiple

containers to run concurrently on the same machine without interference. To execute containers, Docker relies on the Docker Engine, a runtime that operates atop the host operating system, enabling containerized applications to run seamlessly across various infrastructures.

- Docker Images: A Docker image is a read-only template that includes all the elements needed to run an application within a container, such as libraries, configuration files, and system tools. These images serve as the blueprint for containers, with each image instantiated into a container by the Docker Engine.

Docker employs a client-server architecture to manage container operations and it can be observed in Fig. 2.25. The core server component, known as the Docker Daemon, is responsible for building, running, and distributing Docker containers. Communication between the Docker Daemon and the Docker Client is facilitated through a RESTful API over UNIX sockets. For the storage and distribution of Docker images, Docker utilizes registries—central repositories where images are stored and retrieved. The Docker Daemon interacts with these Docker Registries to fetch and store images as needed.

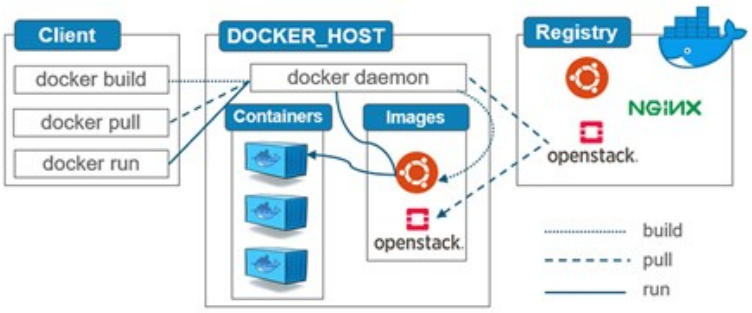


Figure 2.25: Docker Architecture.

Introduction to Kubernetes

Kubernetes [42] (K8s) is an open-source platform designed to automate the deployment, scaling, and management of containerized applications. As a leading container orchestrator,

Kubernetes significantly enhances distributed systems by offering robust functionalities that improve efficiency and reliability.

One of Kubernetes' primary features is load balancing. It exposes applications through Services, assigning DNS names or IP addresses to ensure even distribution of network traffic across containers. Additionally, Kubernetes supports storage management by allowing the mounting of volumes from various storage solutions, including local storage, cloud providers, and Network File System (NFS). A core strength of Kubernetes lies in its ability to maintain the desired state of deployments. Administrators define the desired state using YAML or JSON configuration files, specifying parameters such as the number of running containers and resource limits for CPU and RAM. Kubernetes continuously monitors the actual state of the cluster and takes necessary actions to align it with the desired state, ensuring consistency. Health monitoring is integral to maintaining system stability. Kubernetes continuously checks the health status of containers, automatically restarting or replacing those that become unhealthy. Furthermore, Kubernetes streamlines configuration and secret management, enabling users to handle application settings and sensitive information, such as passwords and SSH keys, without rebuilding container images.

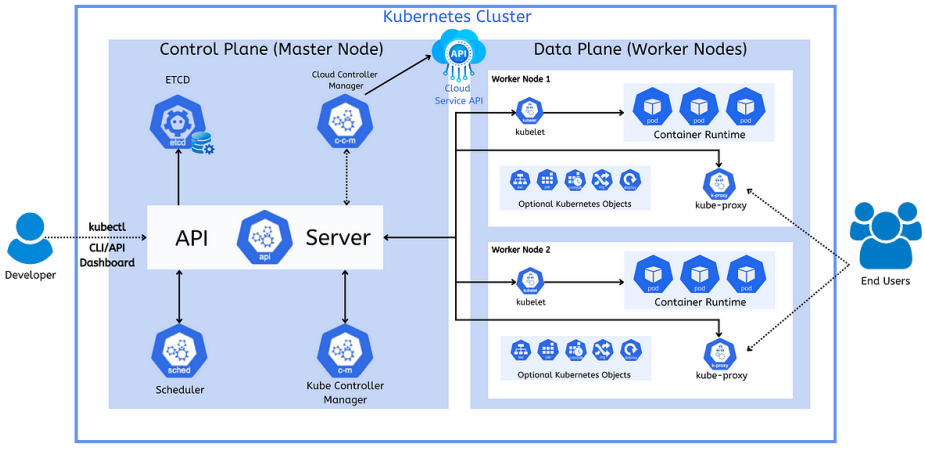


Figure 2.26: Kubernetes Architecture

Kubernetes clusters as illustrated in Fig. 2.26, are composed of nodes that host containerized applications within pods, the smallest deployable units in the system. Each node runs essential components, including the Kubelet, which ensures containers within pods are running and healthy, the Kube-proxy, which manages network rules to facilitate communication

between pods internally and externally, and the Container Runtime, which executes containers using technologies such as Docker, containerd, or CRI-O.

The Control Plane oversees the cluster's desired state, ensuring it aligns with the actual state. Typically hosted on dedicated control plane nodes, it comprises several key components:

- **kube-apiserver**: Serves the Kubernetes API, acting as the primary interface for cluster interactions through tools like *kubectl* or the Kubernetes Dashboard.
- **etcd**: A reliable, distributed key-value store that maintains cluster data, ensuring consistency and availability.
- **kube-controller-manager**: Runs controllers that regulate the cluster's state, ensuring desired configurations are met.
- **kube-scheduler**: Assigns pods to nodes based on resource availability and policy constraints.
- **cloud-controller-manager**: Integrates the cluster with cloud provider services, managing cloud-specific controllers.

Kubernetes objects define the desired state for both applications and infrastructure, described using YAML or JSON. These objects include:

- **Pod**: The smallest deployable unit, typically containing one container with a unique IP address within the cluster.
- **ReplicaSet**: Ensures a specified number of pod replicas are running, facilitating scaling operations.
- **Deployment**: Manages ReplicaSets, enabling seamless rolling updates and rollbacks for application versions.
- **Service**: Provides stable endpoints for sets of pods, ensuring reliable network communication despite pod IP changes.
- **StatefulSet**: Manages stateful applications by maintaining consistent identities and storage for each pod.

- **PersistentVolume (PV)** and **PersistentVolumeClaim (PVC)**: Both manage storage resources, with PVs representing storage and PVCs acting as requests for specific storage requirements.

Finally, networking is fundamental to Kubernetes, enabling seamless communication within the cluster through various mechanisms. Containers within the same pod communicate via a shared network bridge without requiring Network Address Translation (NAT), facilitating efficient intra-pod communication. Kubernetes utilizes the node's root network and Linux bridge to route packets efficiently for communication between pods on the same node. When pods reside on different nodes, routing tables direct traffic between node-specific network bridges, ensuring reliable data transfer across the cluster.

KubeVirt

By default, Kubernetes does not provide native support for managing virtualized technologies such as VMs. This limitation is addressed by KubeVirt [43], an extension for Kubernetes that enables the management of libvirt-based virtual machines within the Kubernetes ecosystem. Rather than segregating containers and VMs, KubeVirt integrates VMs as container workloads, allowing them to be managed alongside traditional containerized applications. The hybrid availability of both containers and VMs is particularly advantageous for edge solutions in modern cellular networks, where the lifecycle, scaling, and migration of edge services can be efficiently managed using KubeVirt's unified API. For instance, deploying a MEC host, including the MEC agent and MEC application, can be done on a VM managed by KubeVirt's API.

The architecture of KubeVirt as illustrated in Fig. 2.27 comprises several key components. The *virt-api-server* serves as the interface exposing KubeVirt's API, handling updates related to virtualization through custom resource definitions (CRDs) and managing the validation and defaulting of VM configurations. The *virt-controller* oversees the pods associated with VMs, monitoring their status. Similar to Kubernetes' kubelet, the *virt-handler* runs on each worker node, continuously monitoring the state of VMs to maintain the desired state. The *virt-launcher* is responsible for managing the namespaces that host VMs, initiating VM instances by passing their CRD objects based on signals from the virt-handler. Within each VM pod, *libvirtd* operates to manage the lifecycle of the VM process, utilizing containerized libvirtd and QEMU technologies to deploy and run VMs effectively. Moreover, KubeVirt

allows the management of VMs similarly to containers, including the ability to perform live migrations through standard `kubectl` commands. Live migration is initiated by submitting a `VirtualMachineInstanceMigration` object to the cluster, specifying the VM to be migrated. Additionally, migration parameters such as reserved bandwidth can be configured via Kubernetes ConfigMaps. KubeVirt employs pre-copy techniques by-default for VM live migrations, which help minimize downtime during the migration process.

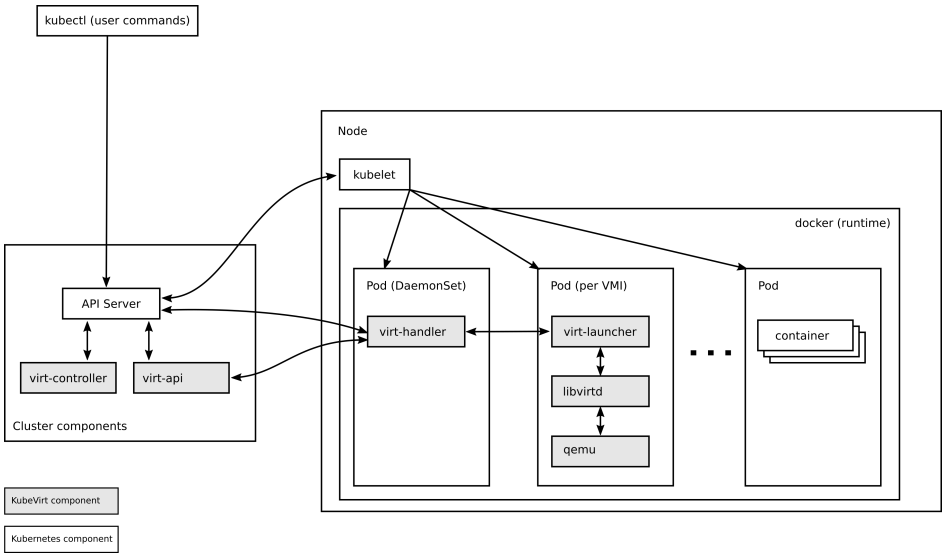


Figure 2.27: KubeVirt Architecture.

Kubeflow

Kubeflow [44] is an open-source platform engineered to simplify the deployment, orchestration, and management of ML workflows on Kubernetes. It emerged from the necessity to address the complexities involved in deploying scalable ML models. By abstracting the underlying infrastructure, Kubeflow allows users to concentrate on developing and deploying ML models without being bogged down by system management intricacies.

The primary objective of Kubeflow is to support the entire machine learning lifecycle, including data preparation, model training, hyperparameter tuning, deployment, and monitoring. It integrates various ML frameworks and tools into a unified platform, ensuring seamless interoperability and scalability. At the core of Kubeflow's architecture is the Kubernetes cluster. Kubernetes ensures that ML workloads run efficiently and reliably. Building on this foundation, Kubeflow introduces specialized components like Kubeflow Pipelines, which facilitate the design, deployment, and management of end-to-end ML workflows. These

pipelines offer a graphical interface for creating complex workflows that include data processing, model training, evaluation, and deployment stages, defined using a domain-specific language (DSL) that promotes reproducibility and version control.

Model serving and deployment are streamlined through Kubeflow's KFServing component, which offers a standardized interface for deploying and managing ML models in production environments. KFServing supports features such as autoscaling, canary deployments, and A/B testing, ensuring that models are served with high availability and minimal latency to meet real-time inference requirements. Additionally, Katib, Kubeflow's hyperparameter tuning tool, automates the search for optimal hyperparameters using various algorithms, enhancing model performance through iterative experimentation and refinement. A high-level overview of the ML lifecycle within Kubeflow ecosystem is depicted in Fig. 2.28.

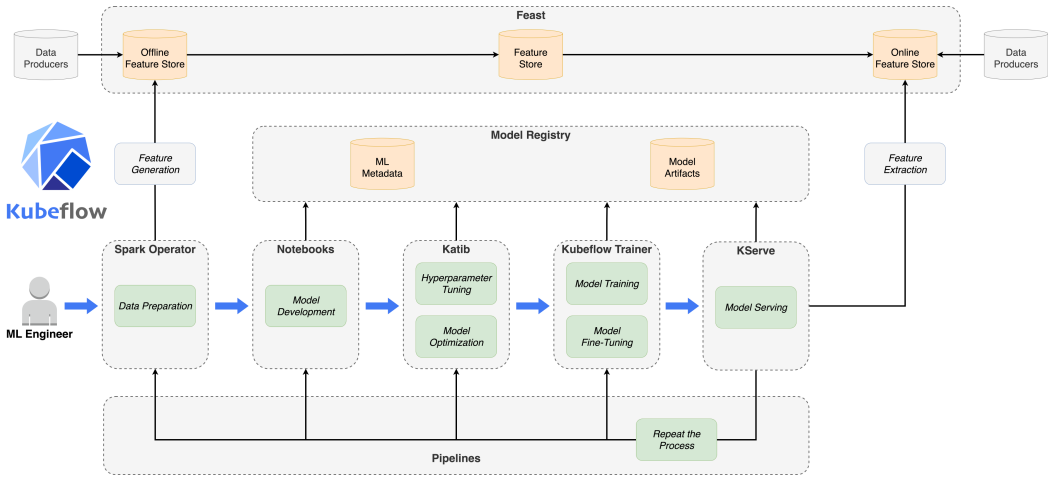


Figure 2.28: KubeFlow ML lifecycle.

Chapter 3

Mobility Aware Edge Service Migration for 6G Networks

3.1 Introduction

The evolution of the telecommunications infrastructure to the 5th generation has enabled key functionalities, allowing the flexible management, deployment, and subsequent chaining of network components as software network functions. Such approaches have been enabled through the wide application of softwarization for the different network functions, applied even for the Radio Access Network (RAN), and the disaggregation of previous monolithic components (e.g. the cellular core network and the RAN base stations) to separate functions. In this manner, the network can be dynamically and even autonomously adjusted [45] in an end-to-end manner, based on the actual load that it is experiencing, allowing the transition to self-managed and organized 6G networks.

Cloud-native approaches for network functions in the 5G context have been embraced by the community, as they allow flexible management, reconfiguration and monitoring of the network in an end-to-end manner. As low latency access is needed in the 5G and beyond networks for serving ultra Reliable Low Latency Communications, Multi-Access Edge Computing (MEC) needs to be integrated in the overall architecture. As the user moves among different RANs, the latency of accessing the service needs to be preserved for providing users with a seamless experience. To accomplish such behavior, migrations of the hosted services are needed, though not fully compatible with the cloud-native approach, and placing them closer to the network access point of the user.

In this chapter, we experiment with a cloud-native end-to-end network, enhanced with Follow-me MEC functionalities. Heterogeneous access is provided at the RAN level, using disaggregated base stations, and MEC is integrated on the fronthaul of the network, ensuring low-latency access to services. The entire network is instantiated in a cloud-native manner, using a widely adopted container orchestration solution. Thus, the approach is a fully cloud-native Follow-me MEC system, that self-organizes and migrates the network to the edge server that ensures the minimum possible latency for accessing the service. The network is extended with multiple technologies for the RAN, allowing end-users to reach services located at the far-edge of the network. Our results show that the scheme is able to provide low latency access to the hosted services, while the UE remains agnostic of the entire process and without any drops of the already established connections.

3.2 Related Work

The containerization of RAN functions has gained a lot of attention lately, towards providing the RAN as a Service (RANaaS) in cloud-deployments [46]. Such efforts are empowered by the wide softwarization of the base station stack [38], supported by the disaggregation of the stack to achieve real-time signal processing in a cloudified environment. Towards providing the network under the RANaaS paradigm, the deployed network functions need to be appropriately managed, chained, and monitored. To this end, authors in [47] provide their approach in managing the softwarized disaggregated stack of a base station in RedHat OpenShift. Authors in [48] consider the problem of slicing in such softwarized cloud-native networks and provide a cloud-native approach for network slicing. The proposed approach advances the architectural vision of the mobile network from a network of entities to a network of capabilities, upon which slicing is employed.

Although cloud-native approaches can assist in the overall flexibility for managing the deployed networks, 5G and beyond network advances are enabled through the integration of novel features in the networking stack. For example, for supporting ultra-Reliable Low-Latency Communications (uRLLC), the wide deployment of edge infrastructure is needed. To this end, authors in [49], [50] and [51] pinpoint the important role that MEC can play in reducing the latency time to access services over the network and all the available placements for the services with respect to the different cellular network components. In these works, the

placements are considered at the best case just after the base station component, or after the core network, and placing the core network close to the edge. In previous works [52] and [53], we introduce a novel placement of the MEC services on the fronthaul of the network when considering disaggregated base station setups. Such configuration can impact the latency times for accessing the services, by removing the processing costs of the higher layers of the base station stack, and transmissions of data to the core network. Moreover, the access network is augmented with heterogeneous links, and can therefore support different network access schemes, with varying access times to the services placed on the fronthaul.

Although MEC can drastically reduce latency for certain types of services that are deployed along the network edge, it does not in principle consider the mobility of the end-users. To cope with this problem, and keep the latency times low, the Follow-me MEC approach has been proposed. In [54], authors apply such approach in a vehicular environment. They introduce their algorithm for migrating the services to other hosts, and consider an SDN-based control plane for managing the network substrate. In [55], a distributed storage is used for synchronizing states between different MEC servers, ensuring service continuity when switching to another server. Authors in [56], formulate the problem of determining the new hosts for migrating the services in such an environment and propose a distributed approximation scheme with reduced time complexity.

In this chapter, we progress beyond existing works by building an entirely cloud-native end-to-end network, with Multi-access Edge Computing functionalities. The network is further extended to support migration of the services, subject to the quality that is measured from the operator side. The end-to-end network, MEC and migration capabilities, and monitoring framework are managed through the same orchestrator solution. In the following section, we present our low-level system setup, and how we enable live migration of the containerized MEC services.

3.3 System Architecture

Towards deploying the entire network in a cloud-native manner, we employ the Kubernetes framework. Our implementation comprises a heterogeneous 5G disaggregated network with novel MEC functionalities, allowing services to be placed directly over the fronthaul of

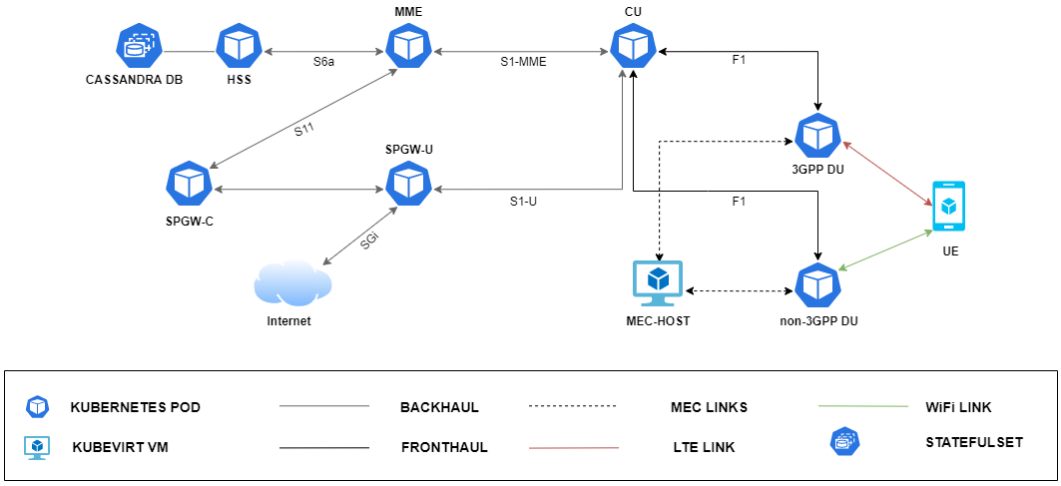


Figure 3.1: The deployment of Heterogeneous MEC-functional 5G Network on Kubernetes.

the network, being completely managed and deployed through the Kubernetes framework. Figure 3.1 illustrates the end-to-end system architecture that enables the cloud-native RAN and MEC setup. Below we list the main components of the solution, that enable the migration of the MEC services to new hosts, in a seamless manner.

3.3.1 Management and deployment of the network functions

The Kubernetes framework is employed for managing and deploying the end-to-end network and MEC services, as well as for their real-time monitoring. The control-plane node (master) is running as a Virtual Machine (VM), managing different worker nodes that host the actual network services. The architecture components end up being Kubernetes pods, which host containerized instances based on an implementation of OpenAirInterface platform [38] that has been developed in our prior research works [57] [52]. All these instances together comprise an end-to-end containerized and disaggregated heterogeneous 5G network, as detailed further in the subsections below. On top of this architecture, we integrated the functionality of MEC, deployed as virtual machines, and managed by the overall system as container workloads, with the assistance of the KubeVirt plugin [43] which it's architecture analyzed in chapter 2.4.3. The choice of hosting the MEC services as VMs rather than containers gives us the advantage of providing live migration of the services, even stateful ones, without any drops of already established connections. In this way, we can manage VMs as we could manage containers and take advantage of VMs Live Migration by executing control plane commands.

Regarding the selection of Kubernetes Network, we deployed the Flannel CNI (Container Network Interface) plugin to the cluster. In addition, we used Multus CNI to have more than one default pod IP, extending the interfaces of the pods for multiple connectivity between the 5G network components. To accomplish a stateful Live Migration of the VMs, network bridge interfaces are used on the worker nodes. On these bridge interfaces, the VMs attached their own static IPs, providing Layer 2 connectivity between them.

3.3.2 RAN Functions and MEC

Regarding the actual implementation of the disaggregated network, we employ the implementation with the functional split taking place in the higher OSI stack layer 2, between Packet Data Convergence Protocol (PDCP) and Radio Link Control (RLC) layers of the base station. The upper layers play the role of the Central Unit (CU) and the lower layers the Distributed Unit (DU), which performs the actual transmissions over the air. Multiple DUs can belong to a single CU and communication between them is based on F1 Application Protocol (F1AP) via the F1 interface. This allows us to have different transmission paths, to serve a UE at the same time. The implementation running inside the pods also supports the integration of non-3GPP DUs (e.g. a WiFi DU, as described in detail in [57]), by properly handling of the transmitted data to/from the CU. Since DUs need to have an appropriate RF frontend to perform the transmissions over the air, we employ volumeMounts for the 3GPP DU, allowing the USB device to be handled by the container hosting the service. Regarding the non-3GPP DUs (WiFI), we run the pod with the *host network* enabled. This allows the container to have direct access to all the network interfaces of the hosting worker, and have direct access to the WiFi chipset that is used to run the WiFi network. The WiFi configuration utilizes 802.11n channels.

3.3.3 Follow-me MEC extensions

In this section, we provide the details for the operation of the MEC over the fronthaul functionality, and how it has been extended to support the Follow-me MEC functionality.

In order to incorporate services over the fronthaul, we need the appropriate interfaces between the DUs and the MEC platform hosting services/applications. In [52], we developed a protocol for DU to MEC communication and introduced a *MEC Agent* component. The agent generates and exchanges the appropriate messages with the DUs, and receives and

delivers the respective payload destined for services hosted at the MEC site. The solution is similar to the *bump-in-the-wire* method by ETSI [51], but traffic interception is taking place on the fronthaul (between the CU and DUs) rather than the backhaul network. The MEC service can also select the technology through which each UE is served in a per-packet basis, enabling the dynamic selection of the links per UE from the MEC's perspective.

Our Follow-me implementation scheme relies on the management of an autonomous single MEC site with the help of the Kubernetes framework. The MEC site is a VM instance delivered as container workload defined by the KubeVirt API, integrating the functionalities of the *MEC Agent* and the hosted service. The placement of the MEC site is located on the fronthaul enabling the lowest latency between the end-user and the services. The architecture components of MEC site can be seen in Figure 3.2. MEC Agent manages the packets going to/from the MEC services, by communicating directly with the DUs, allowing the provisioning of services directly from the fronthaul of the cellular network. The agent holds a book-keeping process for mapping each RNTI value of each UE. Based on this RNTI information, the appropriate requests are made between DU-MEC and vice versa.

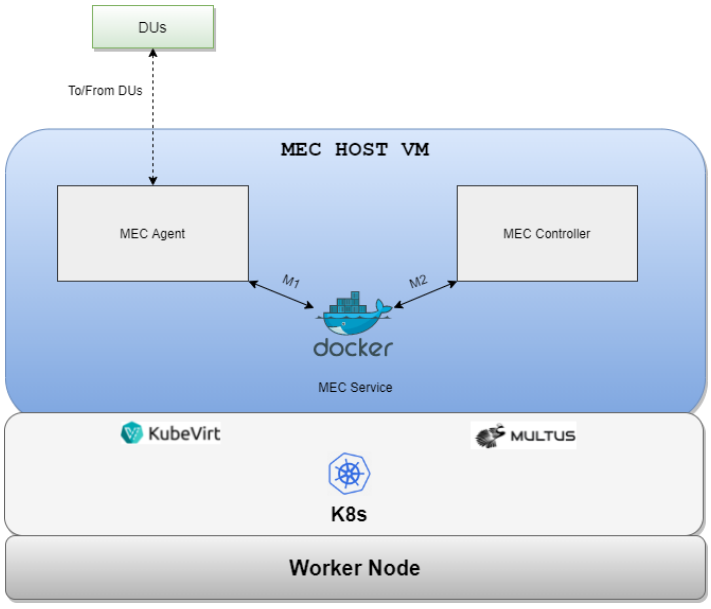


Figure 3.2: MEC Host Architecture.

The live migration of services is triggered by the MEC controller. MEC controller operates internally on the VM and can execute migrate commands as it has remote access to the

Kubernetes cluster API and monitoring tools, and supports RAT (Radio Access Technology) switch functions by sending a signal to MEC Agent to change the transmission path of the MEC service through 3GPP and non-3GPP technologies as is illustrated in Fig. 3.3.

The MEC service is a docker application that is attached by two macvlan interfaces M1 and M2. The M1 interface connects the MEC service with the MEC agent and through this interface passes all the traffic between the service and the end-user. On the other hand, the M2 interface connects the MEC service with the MEC controller. Through this interface, all the traffic related to the monitoring of the quality of the connection between the service and the end-user is transmitted. More specifically, the connection between the MEC controller and the MEC application is based on server-client communication.

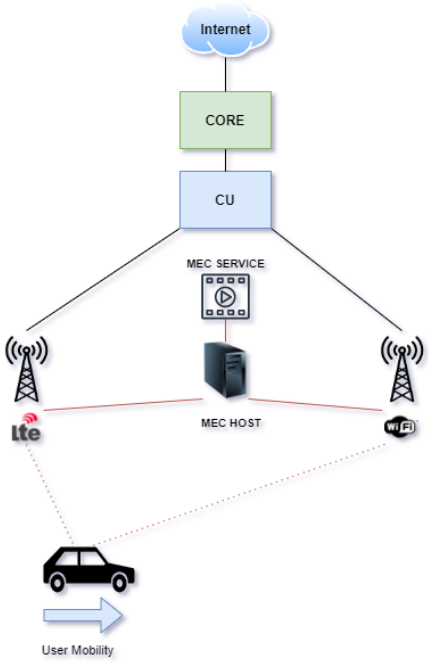


Figure 3.3: MEC traffic passed on dual technology DU's.

The MEC service runs the server-side of the communication, as described in the algorithm 1. In essence, it gathers information about the quality of the link it has with the end-user. This information is mainly related to the Round Trip Time (RTT), which is averaged from the last 10 measurements, using a sliding window approach. Along with the RTT average, the number of packets and the packet loss rate are measured and all of these statistics are stored in a dictionary. This dictionary is then sent to the client which runs on the MEC controller

and whose operation is described in the algorithm 2. The MEC controller after receiving the dictionary constantly monitors if the RTT average exceeds the RTT threshold which is defined depending on the type of application. In case the RTT average exceeds the threshold, then the controller makes a RAT switch by switching the transmission path from LTE DU to WiFi DU demonstrated by Fig. 3.4. If the delay is still high then the controller live migrates the MEC Host to a targeted worker that is closer to the UE as it is observed in Fig. 3.5. After waiting for the average migration time to pass, which is updated at the end of each migration after being parsed by the log files, it switches back to the LTE DU. In the meantime, the user exchanges MEC data via WiFi DU, experiencing a seamless migration experience.

Using the aforementioned approach, the costs of migrations are kept low, as service migration incurs additional operation costs such as usage of the expensive wide-area-network (WAN) bandwidth and system energy consumption [58]. At the same time, we also take advantage of the benefits of a heterogeneous 5G network utilizing all the transmission paths that are available with the sole purpose of reducing UE service access latency.

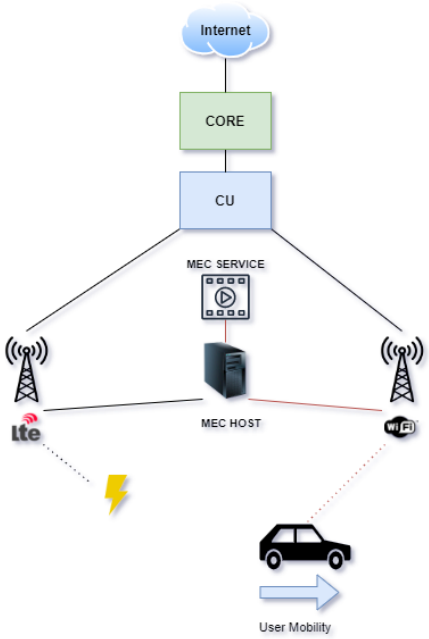


Figure 3.4: Radio Access Technology switch.

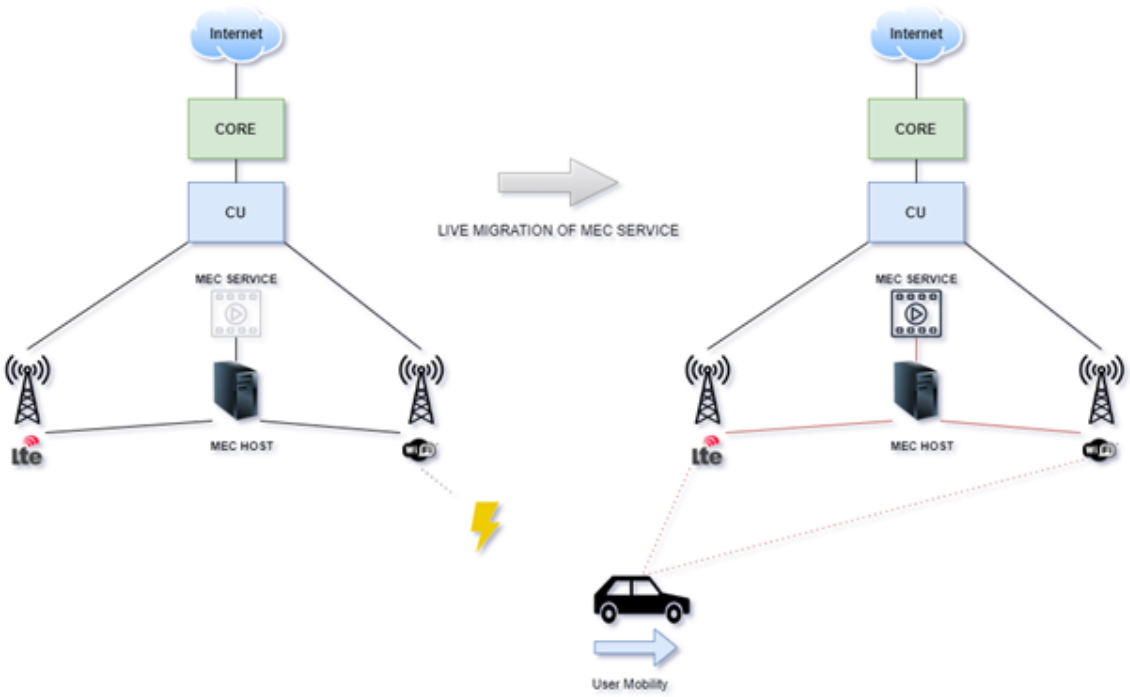


Figure 3.5: Live Migration of MEC service.

Algorithm 1: Follow-Me procedure [Server]

Function `follow_me_server` $(mec\_ip, port, ue\_ip, num\_packets)$ **:**
    $client\_socket =$ `init_server_socket`(mec_ip, mec_port);
    while $True$ **do**
        ping_results = `get_ping_results`(ue_ip, num_packets);
        $ping\_statistics = ping\_results$.`parse`().`as_dict`();
        $msg = ping\_statistics$.`serialize`();
        $client\_socket$.`send`(msg)
    end
End Function

3.4 Evaluation

For the deployment of the heterogeneous containerized 5G network with MEC functionalities, we used the NITOS testbed [37]. NITOS is a heterogeneous testbed located in the premises of University of Thessaly, Greece. The availability of wireless devices (WiFi, Software Defined Radios, UE terminals) suits our experimentation needs for evaluating our solution.

For the experimental evaluation of the cloud-native Follow-me MEC approach, we used four NITOS nodes as Kubernetes workers, while the control-plane node was running on a separate VM on the NITOS cloud. One of the worker nodes was used to deploy the disaggre-

Algorithm 2: Follow-Me procedure [Client]

```
Function follow_me_client(mec_ip, port):
    client_socket = server_connect(mec_ip, mec_port);
    avg_mig_time = init_avg_mig_time();
    rat_switch = False;
    while True do
        msg = client_socket.recv();
        ping_statistics = msg.deserialize();
        if ping_statistics.rtt_avg > rtt_threshold then
            if rat_switch == False then
                switch_to_wifi_du();
                rat_switch = True;
            else
                mec_host = closest_worker();
                kubectl.live_migrate(mec_host);
                sleep(avg_mig_time);
                avg_mig_time = get_avg_mig_time();
                switch_to_lte_du();
                rat_switch = False;
            end
        end
    end
End Function
```

gated Core Network and the Central Unit of our communication scheme. The Core Network is running a disaggregated instance of the OpenAirInterface Core Network, featuring Control/User Plane Separation (CUPS) functionality. This breaks down to hosting five different containers for the Core Network as follows: 1) a Cassandra based database, 2) a Home Subscriber Service (HSS), 3) a Mobility Management Entity (MME), and 4) a control plane Service/PDN Gateway (SPGW-C) and the respective user plane service (SPGW-U). The CU component hosted at the same worker node is integrating the PDCP and above layers, as well as the interface towards the Core Network. On the second worker node, the fronthaul components were deployed, including the 3GPP and non-3GPP DUs, as well as the MEC site. The MEC site is a VM managed through Kubernetes with the KubeVirt add-on, hosting the *MEC Agent* that facilitates the interaction with the DUs, the *MEC Controller* that selects the appropriate DU from the MEC side, and the actual MEC service that is provided to the UEs of the network. The third node is used for performing the live migration of the MEC site

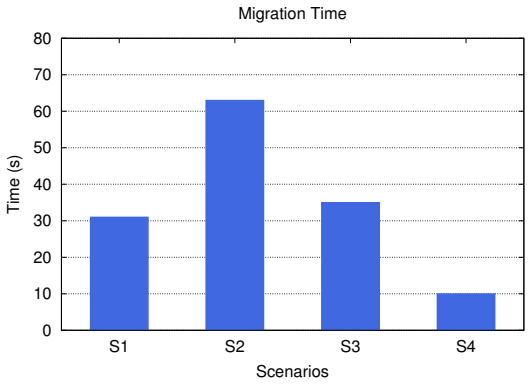


Figure 3.6: Migration Time for each scenario.

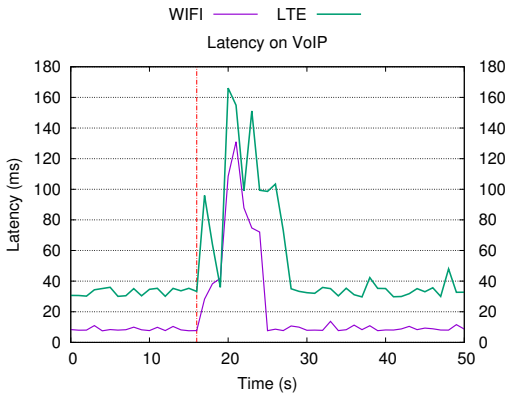


Figure 3.7: Latency on Fronthaul (VoIP application); red line denotes when the migration takes place.

Figure 3.8: Experimental evaluation of the Follow-me MEC system for different scenarios.

during the experiment process. A separate node is used as a multi-homed UE for connecting concurrently to the 3GPP and the non-3GPP DUs.

Table 3.1: Benchmark Characteristics (in ms)

	LTE to MEC-APP	**WiFi to MEC-APP**	**LTE to EPC**	**WiFi to EPC**	**LTE to EPC** (20ms)	**WiFi to EPC** (20ms)
Avg. RTT	25.6	5.28	27.9	5.88	46.03	26.08
Min. RTT	18.76	3.09	22.04	3.21	45.4	25.2
Max. RTT	32.3	12.8	40.8	13.4	54.07	34.9

For the evaluation part of our MEC deployment, we focus on measuring the overall latency for accessing the MEC services. The measurements are based on the latency between the multihomed UE (connected to LTE and WiFi DU) and the MEC service which is deployed either to the fronthaul or to the core network. We noticed that the latency measurements of MEC services deployed on the fronthaul are slightly better than the MEC services deployed on the core network, as illustrated in Table 3.1, considering that latency is about the half of the measured RTT. This is because the core network container instances run on a testbed Node which is relatively close to a testbed Node that the fronthaul container instances run. In real-world scenarios, the core network is usually located several kilometers away from the RAN

components, thus inducing extra delays. To emulate real-world scenarios, we tuned the delay on the link between the CU and the Core Network by injecting 20ms delay for all the traffic, by using the Linux *tc-netem* package. In addition, we can conclude that WiFi outperforms LTE for the cases of latency as shown in figure 3.7. The latency measurements results were conducted during the exchange of VoIP packets between the end-user and the MEC service through the SIPp application [59], that uses Session Initiation Protocol (SIP) to transfer VoIP packets.

To test the functionality of our follow-me implementation, we used 2 different MEC services. One was a real-time message exchange application, based on a TCP/IP socket for the communication of different clients, and the second one was using an application that generates Session Initiation Protocol (SIP) traffic for transfer VoIP packets. As our solution realizes a Follow-me MEC service, we target in evaluating the migration time needed for transferring the MEC service to a new host. Therefore, to measure the migration time, we created and evaluated the following scenarios:

- **S1**: VM includes MEC Agent and MEC Controller (Does not include MEC service)
- **S2**: The VM includes MEC Agent, MEC Controller and SIPp as MEC service.
- **S3**: The VM includes MEC Agent, MEC Controller and real-time messaging as MEC service.
- **S4**: A Fedora VM without hosting any services, instantiated through our framework, used as a reference for our measurements

Figure 3.6 shows our experimental results with respect to the migration time of live migration of the VM for each of these scenarios. In all cases, the migration throughput was measured as 64 MiB/s. It is worth noting that to replicate a large number of VM workloads, we used containerDisk ephemeral storage and not shared storage. This means that during migration, along with the memory pages, disk blocks are copied from the source to the destination VM, contributing to the increase of migration time. For all the cases, the connection from the UE to the services hosted in the MEC platform remains uninterrupted, and only is down for a few milliseconds, when copying the dirty pages from the RAM storage of the initial VM to the target migrated VM. This can be observed in Figure 3.7, where we measure the latency between the UE and the VoIP server that is migrated to a new host, over both access

technologies. The latency is uninterrupted, providing an entirely seamless experience to the UE, though some spikes in the overall latency are observed during the migration process. It is worth noting that such spikes are observed only for stateful transport protocols, as it takes some time for the transport layer to adapt to the new location of the service. For stateless transport, such latency times are significantly lower and almost negligible.

3.5 Conclusion

In this chapter, we presented our approach in developing a fully cloud-native Follow-me MEC scheme using open source platforms. We used a disaggregated heterogeneous base station, with novel placement of the MEC services on the fronthaul, which proven to be very beneficial for latency times. All the components were containerized and managed through the Kubernetes framework. For performing live migrations of the service, we integrated it as a Virtual Machine to the Kubernetes framework, using the KubeVirt add-on. Our results denote that the system is able to perform live migrations, as the latency of the link is deteriorating, and switch technologies on the fly, allowing a multihomed UE to experience seamless low-latency access to the MEC service.

While these results showcase effective reactivity, proactive approaches leveraging predictive mechanisms could further enhance system responsiveness and performance. In the following chapter 4, we build upon these findings by incorporating proactive migration strategies using Deep Reinforcement Learning (DRL), which anticipates service relocation needs based on user mobility and edge-server workload, ensuring even more robust and adaptive MEC service continuity.

Chapter 4

Deep Reinforcement Learning based Service Migration for 6G Networks

4.1 Introduction

The 5G architecture inherently facilitates MEC deployments, especially when the UPF is placed at the network edge. Combining edge-deployed services with close proximity to the UPF minimizes latency by reducing the need to route traffic over long distances to centralized data centers, typically hosting the 5G Core Network (5GCN). Consequently, MEC deployments must inherently accommodate user mobility, seamlessly migrating services to maintain low latency access throughout client movements. Network Functions Virtualization (NFV) further supports this mobility by decoupling network services and functions from underlying hardware. This decoupling allows both core network functions (e.g., UPF) and user services (e.g., VoIP, Video on Demand) to be flexibly hosted as microservices, facilitating migration across edge nodes to follow user mobility dynamically.

Incorporating AI/ML methodologies further enhances service migration capabilities. Predictive algorithms proactively manage service placement and load distribution, ensuring optimal utilization of resources while consistently maintaining user experience standards.

Building upon the cloud-native Follow-me MEC approach presented in Chapter 3, this chapter extends the architecture by integrating proactive migration strategies based on Deep Reinforcement Learning (DRL) techniques. While the previous approach successfully preserved low-latency access to MEC services through reactive migrations, proactive mechanisms that anticipate user mobility patterns and edge-server workloads can significantly en-

hance the QoS and ensure seamless user experiences.

To achieve this, we introduce a comprehensive DRL-based framework leveraging our cloud-native infrastructure built upon Kubernetes and KubeVirt. The framework utilizes multi-cell RTT measurements, provided by the LMF, and real-time workload monitoring to make informed and predictive migration decisions. Furthermore, our enhanced framework supports hybrid migrations, enabling seamless transitions of both containerized services and Virtual Machines (VMs). We extend our evaluations to include migrations of critical 5G network functions such as the User Plane Function (UPF), thereby assessing the performance implications and guaranteeing uninterrupted service continuity. Our contributions are summarized as follows:

- To provide a seamless MEC experience to moving users of the network by exploiting our developed edge infrastructure.
- To enable continuous and uninterrupted low-latency access to services deployed on the network edge.
- To model the service migration environment as a Markov Decision Process (MDP) problem, and to design a reward function that incorporates migration cost penalties to guide the decision-making process.
- To implement DRL algorithms on top of our environments and to compare their performance.
- To select the optimal target location for the edge services by taking advantage of the user's localization, utilizing a DRL agent.
- To evaluate and integrate the developed approach in a real 5G edge setup, using realistic mobility patterns and real-world edge workload dataset.

The remainder of this chapter provides detailed insights into our DRL algorithms, hybrid migration mechanisms, and comprehensive experimental evaluations under realistic mobility and workload scenarios.

4.2 Related Work

Through the definition of the 5GCN in a disaggregated manner and executing it using the Service Based Architecture [49], MEC can be truly realized in a low-cost manner, allowing service providers to take advantage of the network edges for providing selected services with low latency. Such applications are of particular interest to the IoT community, as for certain use cases low latency access and edge selection can be beneficial for the services offered over the top. In [60], authors discuss the role of MEC in 5G and IoT, and demonstrate how IoT applications can benefit from a MEC-enabled 5G network with a use case that utilizes MEC to achieve edge intelligence in IoT scenarios. Authors in [61] exploit the Virtual Machine (VM) technology in order to provide migration capabilities in such IoT edge scenarios, while at the same time reducing the loading time of the VM-based application by mangling the transferred files from each edge host. In [62], the authors model the problem of MEC location selection in an IoT environment as a multiattribute decision-making problem, based on SDN and NFV. In this work, the authors are able to reduce the server response time and improve the quality of the user service experience. Specifically to the 5G network model, authors in [63] present a 5G network architecture together with its network management capabilities, complementing MEC with the connectivity service. The authors address different classes of use cases and applications and evaluate their approach in a testbed setup. Subject to client mobility, modeling the best wireless channel association and service placement within the network is not a trivial task [64], especially when trying to meet a minimum Service Level Agreement (SLA) on latency with the end-user. In [65], authors argue on the applicability of MEC to a vehicular environment where services are replicated across different hosts and prove that their approach can prune the end-to-end communication latency. In [66], authors try to develop MEC solutions coupled with user mobility, for the fast relocation of service instances to guarantee the desired QoE. The authors use containers for hosting the services and develop a framework where proactive service replication for stateless applications is exploited to drastically reduce the time of service migration. In [67] and [68], authors explore the Checkpoint/Restore In Userspace (CRIU) technology to migrate containerized services to different hosts subject to client mobility. Although CRIU provides the ability to migrate stateful applications as well, it fails to address different types of protocols supported in the telecommunications network environment, such as the SCTP protocol for the N1/N2 interface between the Access and Mobility Management Function (AMF) and the gNB. In [69],

authors explore the methodologies for handovers and service migrations employing probabilistic and prediction algorithms, using real-world datasets, and evaluating the implemented models. Similarly, in [70] the authors employ statistical and machine learning models to forecast the edge evolution, in order to get the migration decisions. Although these approaches are valid, classical machine learning and deep learning algorithms don't cope with the dynamic nature of edge environments. Moreover, in order for these models to be effective huge datasets are needed. In such dynamic environments, the use of reinforcement learning (RL) may be necessary in order to effectively adapt to changing conditions and make real-time migration decisions. Additionally, RL-based approaches have the added benefit of being able to consider the long-term consequences of migration decisions, rather than simply predicting the next best action. In [71] the authors propose a DRL approach for service migration in (MEC)-enabled vehicular networks in a simulation environment, observing communication delay and migration costs and evaluating the learning ability of the agent. The work reduces the end-to-end latency and migration costs. However, the solution is tested only in a simulation and there is no system architecture or an explanation of the integration of their approach in real-world infrastructures. On the contrary, in work [72], authors employ DRL for determining the bandwidth for service migrations in 5G Networks. They employ DQN and DDPG algorithms in a continuous action space defined as the bandwidth for the corresponding migrations. They evaluate their algorithm in real-edge infrastructure utilizing CRIU technology to migrate the services. Although their solution targets 5G Networks, there is no integration of their approach into a 5G network with the respective interfaces.

In this work, we progress beyond existing literature by using a cloud-native RAN and Core Network, deployed by using a blend of micro-services and VMs, based on the OAI platform. The selection of the different types of virtualization depends on the services (network/edge services) as detailed further below, consisting of either Virtual Network Functions (VNFs) or Containerized Network Functions (CNFs). We blend the approaches of the CRIU-based microservices and VM-based service provisioning, towards reaping the benefits of both worlds in the k8s environment. By taking advantage of the multi-cell RTT feature standardized by the 3rd Generation Partnership Project (3GPP) and the workload cluster measurements, we model our infrastructure as an MDP problem. We define the states and the actions and we design a reward function that targets optimal decisions and incorporates migration cost-aware penalties. On top, we implemented a service migration Deep Q-Network (DQN)

and Deep State-Action-Reward-State-Action (Deep SARSA) agents. To train the agents, we developed a digital-twin simulation environment identical to our real-world setup. Finally, we evaluate the agent's performance in the real edge infrastructure. In the next section, we detail our system architecture and key building blocks.

4.3 System Architecture

Our overall setup consists of a 5G Edge architecture, that is entirely based on the Kubernetes (k8s) framework, enhanced with novel capabilities for service continuity of MEC applications and maintenance of 5G Virtual Network Functions (VNFs). Fig. 4.1 summarizes the end-to-end service-based 5G network that uses hybrid solutions offered by the coexistence of VMs and containers. By default, there is no built-in mechanism in k8s to support the migration of stateful pods between its cluster nodes. Our architecture covers this gap using various technologies that benefit beyond 5G networks as they can contribute to the seamless experience of users regardless of their mobility. Furthermore, we enhance our architecture with a digital twin-driven DRL framework to forecast the edge conditions and to take optimal migration decisions. Below, we analyze the components of our architecture and the diverse technologies, that make up our setup. To evaluate our implementation, we utilized the NITOS testbed [37], a remotely accessible facility located at the University of Thessaly, Greece.

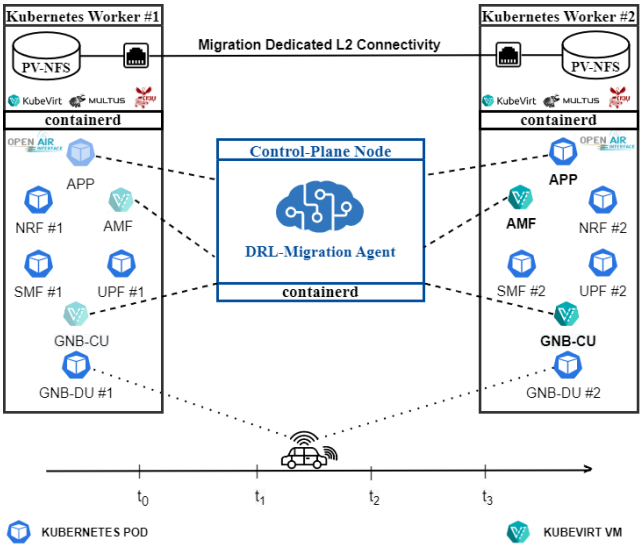


Figure 4.1: The deployment of the live-migration capable 5G Edge Infrastructure on Kubernetes

4.3.1 Architecture of the Edge Infrastructure

Our cluster consists of three k8s workers and one control-plane node. The control-plane node is responsible for monitoring the health of the other nodes as well as the proper operation of the VNFs and services. The remaining nodes host – by default – pods, but also VMs due to the KubeVirt framework that we deployed in our cluster. A pod is the minimal object deployment for a microservice within the k8s environment; it consists of at least one/more containerized services, that are intercommunicating with each other. KubeVirt [43] is an add-on that extends k8s capabilities by delivering VMs as container workloads. The significant addition of KubeVirt to the k8s ecosystem brings an ideal environment for edge solutions as it covers the gap of live migration of services in the k8s by taking advantage of VM live migration. Moreover, it enables us to manage the lifecycle of VMs in the same manner as for the pods via control plane commands.

However, containers can be live migrated too, mainly through the CRIU tool [73] which can restore the checkpointed states of the container to the destination node with the help of the runC container runtime. An important effort to integrate CRIU into k8s was accomplished through the PodMigration-Operator [74, 75], which can migrate a stateful pod across the k8s nodes. Nevertheless, it fails to seamlessly maintain IP/TCP connections since the pod's IP changes on the target host, even if a k8s service assigned with a static IP, routes the traffic to the pods.

We managed to maintain IP/TCP connections without interruptions, by attaching to the pods a secondary interface with the help of the Multus Container Network Interface (CNI). We created static, migration-dedicated MacVLAN interfaces that bind to a host-bridged interface consisting of physical and VLAN interfaces. The pods attach this MacVLAN interface through the ContainerNetworkDefinition with static IPs which are persistent during the live migration. The VMs attach their own static IPs on the same bridged interfaces, providing Layer 2 connectivity between them. With this approach, we were able to incorporate the altered PodMigration-Operator into our architecture.

We apply diskless live migration to both of our virtualized technologies. This allows us to transfer only the memory state of the containers/VMs, which results in fewer memory pages being copied, thus lower migration times. This is achievable as all nodes share a Network File System (NFS), where the NFS server is the control-plane node and the clients are the worker nodes. This NFS system includes the dump files containing the state pods and the images of

the VMs respectively. To this end, the process of live migration in pods can be achieved with the following steps:

1. CRIU snapshots the state of the container on the target pod.
2. The snapshot dump file is exported to the NFS server.
3. A new-cloned pod is created on the target node that restores the source's pod state via the dump file.
4. The target pod is running and the source pod can be removed.

On the other hand, the KubeVirt VMs are importing their disk images through PersistentVolumeClaim (PVC) which is managed by Data Volumes from the Containerized-Data-Importer (CDI) which is a persistent storage management add-on for k8s. In order to perform diskless migrations, these PVCs are distributed to the NFS via the NFS subdir-external provisioner, i.e., an automatic provisioner that supports dynamic provisioning to pods/VMs using the already-existing NFS server. Subsequently, the disk image is always available to the target nodes and only the memory is copied from source to destination.

It is worth mentioning that in both types of migration, the Pre-Copy technique is employed since it has less downtime [76]. By comparing VMs and pods during the live migration, we conclude that pod seems like an ideal solution to deploy the edge services, as it has the least migration time. However, we choose to keep both technologies in our architecture because:

- VM live migrations are more stable and smoother (lower latency spikes) than the pod ones.
- CRIU doesn't support SCTP socket maintenance during live migrations, unlike KubeVirt VMs. This leads to the failure of live migration of 5G CNFs, as almost all of CNFs communicate over the SCTP protocol.

Our final architecture tools and technologies are gathered in table 4.1.

4.3.2 Management & Deployment of Network Functions

For the telecom network, we rely on a multi-slice 5G Core Network provided by the OAI platform, which consists of the following containerized CNFs: 1) Network Repository Function (NRF), 2) Unified Data Repository (UDR), 3) Unified Data Management (UDM), 4)

Table 4.1: Experimental Setup of the Edge Infrastructure

System	Description
Nodes	1- Control Plane Node & 3-Worker Nodes
CPU	Intel-Core i7-3770 @ 3.40 GHz
RAM	32GB
K8s Version	1.19.0
Container Runtime	Containerd
KubeVirt Version	0.45.0
CDI Version	1.43.0
CRIU Version	3.14.0
RUNC Version	1.0.2-dev
K8s NFS Provisioner	NFS Subdir External Provisioner
5G-Core NFs	OAI Multi-Slice Core Network
5G-RAN	OAI RF-Simulator & UERANSIM
5G-UE	OAI NR-UE
5G-SLICE	URLLC

Authentication Server Function (AUSF) 5) Network Slice Selection Function (NSSF), 6) Access and Mobility Management Function (AMF), 7) Session Management Function (SMF), 8) User Plane Function (UPF). In this deployment, there are two Network Slice Selection Assistance Information (S-NSSAIs) configured, therefore two slices: 1) Ultra Reliable Low Latency Communications (URLLC) 2) Massive IoT (MIoT). However, we mainly focus on the URLLC slice. The NSSF, UDR, UDM, AUSF, and AMF are common to all slices, while UPF, SMF, and NRF are unique for each slice.

Likewise, for the RAN, we employ two different RAN simulators: ueransim and rfsimulator [77] corresponding to our two different slices. Specifically, we utilized the disaggregated architecture from the rfsimulator including the CU and DU components, as the result of the gNodeB disaggregation into CU/DU. For the User Equipment (UE), we employed the OAI 5G-NR UE.

Since SCTP socket maintenance is not supported during the live migration of the pods, we decided to nest some of the containerized NFs inside the KubeVirt VMs, in order to be able to

migrate them across the edge nodes. Some of them are UPF, SMF, AMF, and GNB-CU. Their selection was made because most of them are stateful functions and are of significant interest for live migration due to their importance in the control plane proper operation/maintenance and in the QoS that the UPF provides [78]. In opposition, edge services are better to run in pods, so that they can be quickly migrated, as a decrease in QoS in the user plane has a direct impact on end users, while the change in performance of the control plane doesn't directly affect the end user's experience.

4.3.3 Architecture of the DRL Migration Environment

As known, in case of a pod failure, the k8s control-plane node launches a new container in another node to replace the failed one. However, this can cause quite a few problems in an Edge 5G Network. Initially, QoS ceases to exist as there is no service availability. Even worse, crucial CNFs that are essential to the operation of a core network can stop working. In addition, when a network is characterized by its slice, as in the case of URLLC, the new pod that is deployed should be migrated not only to the healthiest node but also to the node that gives the lowest latency with the end-user. To determine the best candidate node we need to observe either the position of the UE or the latency measurements of the neighbor cells/servers along with the load of each node.

As discussed in 2.1.7 chapter, the LMF uses various techniques to measure the location of UE, including using Global Navigation Satellite System (GNSS) signals or using signals from the network itself. However, such techniques face accuracy errors and require good network time synchronization. The 16th release of 3GPP includes support for multi-cell Round Trip Time (RTT) measurements as a new feature in the LMF. Specifically, the UE sends Sound Reference Signal (SRS) requests and receives Position Reference Signal (PRS) responses from multiple Base Stations. We decide to observe the multi-cell RTT measurements for our migration decisions since this method is robust against network time synchronization errors [79]. Additionally, RTT is a more suitable metric for our solution, as it indicates the responsiveness of each cell which sometimes is independent of the UE position (e.g huge cell capacity). Therefore, we end up relying on two metrics for migration decisions. The average *RTT* between the UE and the edge servers/cells, and the *load* of each edge server. We define the average RTT, R_i between the edge nodes and the UEs that the LMF monitors by equation 4.1; where x_i are the average RTT values of the last N transmissions per UE-

node pair. We also define the total *load* L_i of each edge node in the cluster as the uniform degree along multiple dimensions as described by equation 4.2. The variables cpu_i and mem_i are the average utilization of *cpu* and *memory* respectively for the corresponding edge server.

$$R_i = (\frac{1}{N}) \sum_{i=i}^{N} x_i \tag{4.1}$$

$$L_i = \frac{1}{1 - cpu_i} \cdot \frac{1}{1 - mem_i} \tag{4.2}$$

Service migration is a challenging problem due to the dynamic nature of the environment and the complex interactions between the UE, the servers, and the network. Traditional approaches to service migration, such as rule-based or heuristic-based methods, may not be able to adapt to changing conditions or handle complex dynamics effectively. Model-free and policy-based Reinforcement Learning (RL) is well-suited for dynamic environments where the conditions may change over time, such as in a service migration environment where the UE is moving and the loads on the servers may vary. By using RL, the agent can learn an optimal policy for minimizing the RTT between UE and the servers, and for balancing the loads on the servers. This can help to improve the overall performance of the system and provide a better experience for the UE. Traditional RL algorithms such as Q-Learning, use a Q-table to store each state and the corresponding values of all actions (Q-value). However, this cannot scale if the state space expands, since the Q-table will also become larger, resulting in inefficient learning. Deep Reinforcement Learning is particularly effective at adapting to these changing conditions, as it can learn from a large amount of data and can generalize to unseen situations. Moreover, DRL employs a Q-function rather than a Q-table and utilizes deep neural networks (Deep Q-Networks/DQN) that estimate the Q-function, resulting in effective and scalable learning. By taking the aforementioned into account, and by exploiting the edge migration capabilities of our architecture, we designed and implemented a DRL Service Migration framework. The architecture of our solution is illustrated in Fig. 4.2.

We created two identical custom environments, by utilizing the OpenAI Gym platform [80]. The first one is a simulation environment for training purposes, playing the role of a

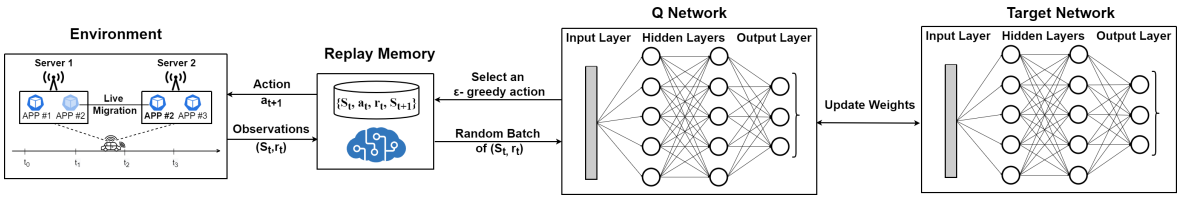


Figure 4.2: Deep Reinforcement Learning Architecture for the Live Migration Environment.

digital twin in the real environment. The second one is for evaluating our solution in a real-world environment. The only difference between them is that the real-world environment employs the real cluster and leverages the migration APIs that we developed in the section 4.3.1. This allows us to safely and efficiently explore a wide range of possible scenarios and actions without damaging real-world systems. Both environments are modeled as a Markov decision process, with the same states *s*, actions *a*, and rewards *r*. The states can be observed by the equation 4.3.

$$s = (N_i, R_1, ...R_N, L_1, ...L_N) \tag{4.3}$$

In this equation, the states are represented as a tuple of the variables; N_i is the Node where the set of user's edge services (pods/VMs) are running and $_N$ is the total number of nodes. In our cluster, we have three edge nodes, thus the states can be redefined as s = (N_i, R_1, R_2, R_3, L_1, L_2, L_3). The action space *a* includes the actions: *Wait* and *Migrate to* N_i *server*. The Migrate action migrates the set of edge services to a specific N_i edge server, thus including as many migration possible actions as the number of edge nodes. The *Wait* action simply means that the agent doesn't migrate the services to any of the candidate nodes at the current time-step. The rewards *r* represent the feedback the RL agent receives after taking action in a given state. We define two local rewards: r_R and r_L.

$$r_R = \frac{R_{min}}{R_i} - \frac{R_i}{R_{max}} + \frac{\min(R_{list})}{R_i} - \frac{R_i}{\max(R_{list})} \tag{4.4}$$

$$r_L = \frac{L_{min}}{L_i} - \frac{L_i}{L_{max}} + \frac{\min(L_{list})}{L_i} - \frac{L_i}{\max(L_{list})} \tag{4.5}$$

Both rewards determine the feedback for each action taken by the agent from the point of view of *RTT* and *load* of the edge servers respectively. Each reward is calculated to a similar respective mathematical formula given by equations 4.4 and 4.5. In these equations, the values R_i and L_i are the current *RTT* and *load* values of the server that the agent migrated or

stayed to. The ranges (R_{min}, R_{max}) and (L_{min}, L_{max}) are *RTT* and *load* Service Level Agreement (SLA) thresholds. Furthermore, the lists R_{list} and L_{list} contain the *RTT* and *load* values for all the candidate nodes. Both rewards are designed to reward/penalize the agent when the migrated server's *RTT* or *load* values are inside/outside SLA thresholds and when the agent selects the most optimal/mediocre server between the candidate nodes. Specifically, the first subtraction of the fractions expresses the ranking of the node in the SLA thresholds, while the second one expresses the ranking of the node among the candidate nodes. In addition, both rewards are migration-cost-aware since they penalize the agent if it migrates the services to less optimal nodes. For example, in the case of *RTT* if the measurements are the following for each node: $(3, 5, 6)_{ms}$ the corresponding reward functions will be: $(0.6, -0.2, -0.6)$ with thresholds defined as $R_{min} = 1$ and $R_{max} = 20$. Although every *RTT* value is far below the R_{max}, the reward functions are negative for the less optimal nodes, preventing the agent from migrating services to them. This results in resource and bandwidth saving, as the agent will try to migrate or stay to the most optimal node, instead to follow an always-migrate policy. However, to better assist the agent to avoid relocating services to the most unhealthy node, we introduce a migration penalty. This penalty is given by equations 4.6 and 4.7.

$$p_R = \begin{cases} +\frac{\text{cost}}{2}, & \text{if } R_i > R_{\max} \text{ and } R_i = \max(R_{list}) \\ 0\,, & \text{otherwise} \end{cases} \tag{4.6}$$

$$p_L = \begin{cases} +\frac{\text{cost}}{2}, & \text{if } L_i > L_{\max} \text{ and } L_i = \max(L_{list}) \\ 0\,, & \text{otherwise} \end{cases} \tag{4.7}$$

Both p_R and p_L, penalize the agent for exceeding the maximum allowed RTT/load, and for selecting the node with the worst rank among candidate nodes. Each penalty amounts to half of the migration cost. This cost is a hyperparameter and in our case, it symbolizes the maximum bandwidth that can be wasted for one service relocation and it's a constant. Finally, the global reward is the sum of the two individual rewards subtracted by the sum of the two individual penalties, as can be observed by equation 4.8. It is worth mentioning that each time the *RTT* or *load* thresholds are exceeded we terminate the episode. This approach has been taken, to guide the agent to not violate the SLA and to address the credit-assignment problem. The credit-assignment problem occurs when the agent receives the reward at the end of each episode without identifying the responsible actions.

$$r = r_R + r_L - (p_R + p_L) \tag{4.8}$$

To evaluate the performance of the proposed DRL solution, we employ real-world scenarios. Since multi-cell RTT measurements-datasets are not yet publicly available, we implemented a realistic mobility scenario. This scenario emulates a part of a real-world 5G commercial topology, located on State Route 111 highway, California U.S. The map topology illustrated in Fig. 4.3 is obtained by the Ookla 5G Map [81]. Precisely, this scenario emulates cars traveling on the given highway in both directions, with speeds varying from 80 to 104.5 km/h with the limit of the highway being 105 km/h. On this route, there are three 5G antennas, with approximately equal distance between them. We assume that the edge servers are located next to the antennas and that we monitor the Muli-Cell RTT measurements through the LMF. The RTT values are linearly proportional to the Euclidean distances between UEs and edge servers/base-stations. Also, the RTT values are affected by the radio interference as random loss, which we generate by adding Additive White Gaussian Noise (AWGN) with a fixed standard deviation per route. The rate at which the RTT changes depends mainly on car speeds and different driving profiles. To generate a large variance that could lead to efficient learning and generalization of the agent, we distribute the variety of speeds uniformly. Subsequently, the RTT values of UE/cell pair change as $\delta_{RTTi} = \frac{d_i}{v_i} + \lambda$, where the d_i is the distance between the UE and the corresponding edge server, vi is the velocity defined by $v_i = uniform(80, 104.5)$ and the random-loss $\lambda = awgn(0.5, 0, 1)$.

To emulate realistic edge workloads, we relied on Google cluster workload traces [82]. This open dataset includes resource requests and usage measurements from Google's Borg cloud clusters, for an entire month. Specifically, we utilize the average *cpu* and *memory* usage from three different machine IDs in the cluster, given by the corresponding equations: $cpu = \frac{\Sigma(U_{\text{cpu}})}{T_{\text{window}}}$ and $mem = \frac{\Sigma(U_{\text{mem}}) \cdot T_{\text{sample}}}{T_{\text{window}}}$. The variables U_{cpu} and U_{mem} are the *cpu* and *memory* usage respectively, while the T_{window} is the measurement window and T_{sample} is the length of the sample. We obtain the *cpu* and *memory* every time-step and we calculate the total *load* per edge server given by equation 4.2. In order to avoid overfitting and to have a large variance to the repetitive load scenario we apply additional Gaussian noise to memory and CPU respectively. The noise is applied each time the scenario is repeated and follows a normal distribution with a mean of 0 and a standard deviation of 1 for both *cpu* and *mem* metrics of each node.

This way we can observe all the states that we defined on tuple 4.3 by emulating mobility and load scenarios that take place in a well-defined real-world topology with realistic load patterns among the edge servers.

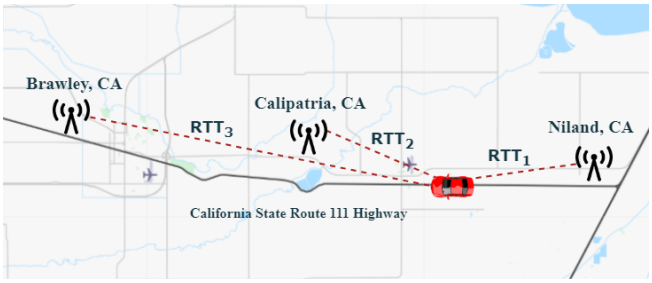


Figure 4.3: Part of a real-world 5G commercial topology located near State Route 111 highway, California U.S.

To learn an optimal policy for this environment using DRL, we utilize a deep Q-network approach, where the DQN agent is trained to predict the expected reward for each action in a given state via the Q Network. The Q network is a neural network and in our case, the Q network is a Multi-Layer Perceptron (MLP). It is responsible for approximating the action-value function $Q(s, a)$ and is updated at each time-step based on the current state and chosen action. In order to stabilize the learning process, we implemented also a target neural network in our system. The target network is identical to the Q network and is used to generate the target values for the Q network updates [83]. The target network is not involved in the training and it is only updated by the Q Network periodically. This results in the reduction of the variance in the learning process and can improve the stability of the system. In addition to the Q network and target network, we employ a replay buffer to store past experiences and sample them during the training process. This helps to decorrelate the experiences and can also improve the stability and sample efficiency of the learning process. In more detail, the $Q(s, a)$ is updated based on experiences in the environment, which are stored in the replay buffer and sampled for learning. At each step, the Q network takes the current state as input and produces a vector of estimates of the action-values for each possible action. The Q network is then updated using gradient descent to minimize the mean squared error between the predicted and target values. The target network is periodically updated to match the weights of the Q network and produces the target values. Then, computes the estimated return of taking the selected action in the current state and the optimal action in the next state via y_i:

$$y_i = r + \gamma \max_{a'} Q_{target}(s', a'; \theta)$$

where r is the reward received after taking action a, s' is the next state, γ is the discount factor that controls the importance of future rewards and θ symbolizes the updated weight parameters. Finally, the Q(s,a) is updated based on the cost function $L(\theta)$ which is the squared difference between target Q and predicted Q:

$$L(\theta) = \mathbb{E}_{s,a,r,s'} \left[(y_i - Q(s, a; \theta))^2 \right]$$

In addition to the DQN algorithm, we also implemented another RL algorithm called SARSA (State-Action-Reward-State-Action). In contrast with DQN, SARSA is an on-policy algorithm as the $Q(s, a)$ is updated based on the current choices of the policy. The SARSA algorithm differs from DQN in the way the target values are computed. Instead of using the maximum expected future reward, SARSA uses the reward and the expected action value of the next state to update the current action value via y'_i:

$$y'_i = r + \gamma Q_{target}(s', a'; \theta)$$

In our implementation, the SARSA agent employs a similar DRL architecture as the DQN with a Q-network (MLP neural network). However, in our case, SARSA doesn't utilize a replay buffer and a target network. This kind of implementation is mentioned by the literature as Deep Sarsa or DSQN [84].

To address the "exploration vs exploitation" problem, we employ the LinearAnnealed-Policy for both algorithms. In this policy, the exploration rate ϵ that controls the probability of selecting a random action is decreased linearly. This reduction rate is controlled by the exploration rate decay d which directs the rate at which ϵ decreases over time. This allows the agent to gradually shift from exploration to exploitation as it learns the optimal actions for a given state. To implement the DQN and SARSA architectures we relied on TensorFlow keras-rl2 python library.

All the aforementioned hyper-parameters and the corresponding values we used for the optimal training are gathered in the table 4.2 after extensive experimentation. The common hyper-parameters of DQN and DSQN algorithms such as Q network, γ, α, ϵ, and d are chosen to have the same values for close comparison.

Table 4.2: Deep Reinforcement Learning Parameters.

Parameter	Value
Deep Q Network	MLP
Deep Q Network depth	2
Hidden layer depth	24
Optimizer	Adam
Activation	ReLU
Target Model Update	20
Replay buffer size	20000
Discount factor γ	0.99
Policy	LinearAnnealedPolicy
Learning rate α	0.001
Exploration rate ϵ	1.0
Exploration rate decay d	0.1
Number of steps	200000

4.4 Evaluation

For the evaluation part of the Edge-Cloud Infrastructure, we initially compared the migration times on KubeVirt VMs and pods in various types of applications: 1) Text Application server, 2) SIPp [59] server, and 3) VLC streaming server. The SIPp application uses Session Initiation Protocol (SIP) to transfer VoIP packets. As illustrated in Fig. 4.4, the migration times are considerably lower in the pods compared to the KubeVirt VMs. However, the migration times of the VMs are generally not prohibitive. Next, we focused on the migration times of VMs that are hosting various NFs including SMF, UPF, AMF, and CU. The operation of the network functions is uninterrupted and the AMF-VM has the longest migration time and this can be observed by Fig. 4.5. Next, we captured the latency and the throughput that the end-user experiences during the interactions with the SIPp server while the server was migrating to other edge nodes, as a pod, and as a VM. The results are displayed in Figs. 4.6 and 4.7, where in both plots, the vertical-dotted line denotes the time that the migration was initiated. Fig.4.6 shows that the VM migration had a smoother impact on the experience of UE in contrast to pod migration which completed much faster (at 47th second), but had

a significantly higher spike in the observed latency. Fig.4.7 shows that the end-user had a seamless experience in terms of throughput in both virtualization technologies. There was only a small imperceptible drop during the migration, which was followed by a small rise. It is worth noting that the service relocated to a node that is closer to the UPF/g-NodeB. Therefore, a small drop in jitter and a small increase in throughput are subsequently observed. To capture both the jitter and throughput in real-time, we utilized the scapy [85] python library.

Toward evaluating the DQN and DSQN agents, we trained both agents for 200.000 steps. Fig. 4.9 illustrates the average reward over the training episodes for the two agents. Both agents were able to learn and improve their performance over the course of the episodes. Moreover, they explored the action space effectively in the beginning, and then switched to exploiting the learned policy as the episodes progressed. Specifically, before the 600th episode both agents were fully exploring the environment. After the vertical line, the agents progressively started exploiting and this is demonstrated by the increase in average reward over the episodes which by the end converged. However, the DQN agent had a better performance, as it reached higher rewards. This denotes that the DQN agent is trained efficiently, as there's a good balance between exploration and exploitation. Additionally, the DQN agent maintained the QoS at higher levels during the phase of exploiting. This is indicated by Fig. 4.10, which displays the increase in the average episode duration. The QoS is increased since we terminate the episodes, each time the RTT or load thresholds are exceeded. This means that the DQN agent took better actions that met the conditions of the SLA. Although sometimes failed to not violate the SLA, due to the fact that the cluster might be overloaded. For the aforementioned reasons, the DQN agent qualifies for the taking of service migration decisions.

We evaluated the DQN agent's performance, in the real-world k8s cluster with the developed migration APIs. Specifically, by taking advantage of TensorFlow's save/load methods we loaded the saved agent's model weights and started testing it in our second evaluation environment. We employ the mobility scenario, by emulating a car that has the max speed on the highway for one round-trip (two-way route). We also apply an unseen heavy edge workload from one specific day of the entire measurement month by leveraging Google's Borg cloud cluster dataset. The OAI-NR-UE interacts with the SIPp server that is packaged as a pod, in order for the agent to achieve the least migration times. The footprint of this experiment is illustrated in Fig. 4.12. The RTT and load thresholds are $R_{max} = 20$ and $L_{max} = 12$

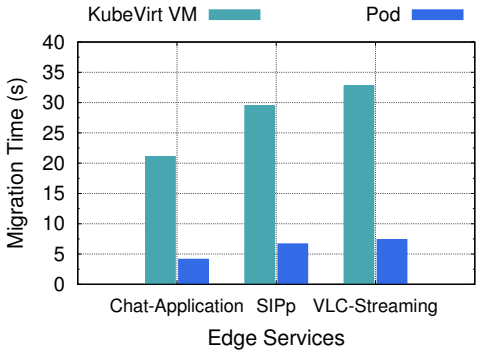


Figure 4.4: Migration time on services: *VM vs Pod*.

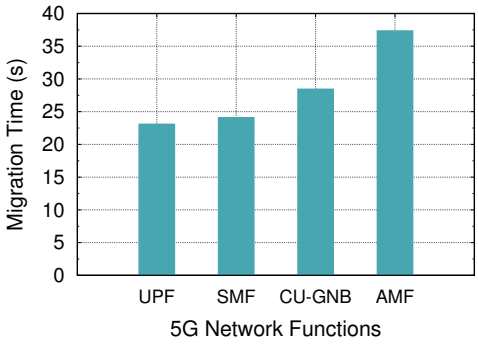


Figure 4.5: Migration time on *NFs* as *VMs*.

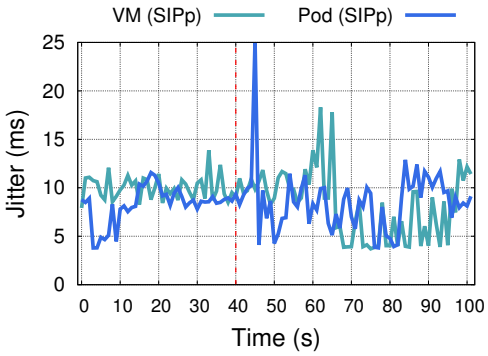


Figure 4.6: End-to-End *Jitter* during migration of services: *VM vs Pod*.

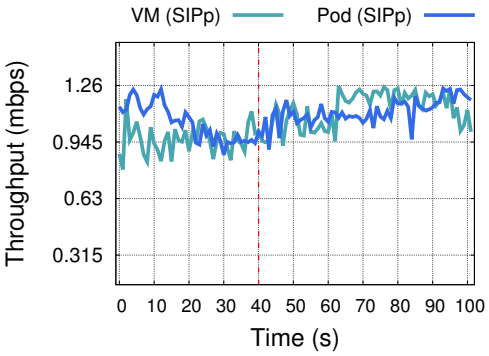


Figure 4.7: End-to-End *Throughput* during migration of services: *VM vs Pod*.

Figure 4.8: Live Migration measurements.

respectively. At the beginning of the experiment, the agent stays on the optimal Node 1. Then at the time points 1 and 2, the agent follows the UE by relocating the service near the end-user (Follow-Me approach), before the RTT threshold is exceeded, and at the same time, schedules it in healthy nodes. Before time point 3, the service is running on Node 3 but as the RTT increases the agent should relocate the service to a better candidate node. However, in that case, every node is out of SLA thresholds, so instead to migrate the service to the overloaded but with satisfactory RTT Node 2, it remains on Node 3 violating the SLA but saving resources. Finally, at time point 3, it relocates the service to Node 1, which falls within the SLA thresholds. This implies that the agent learned the policy successfully, as it proactively relocated the service to the nodes with the optimal RTT and load values and saved resources without performing unnecessary migrations.

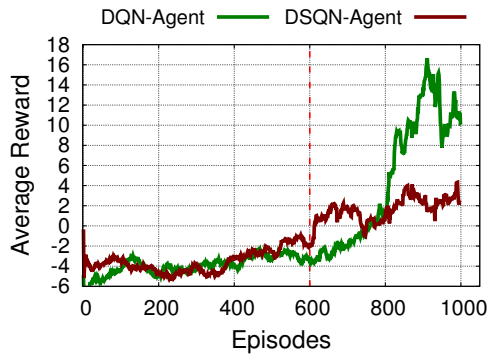


Figure 4.9: Average reward per episode during training.

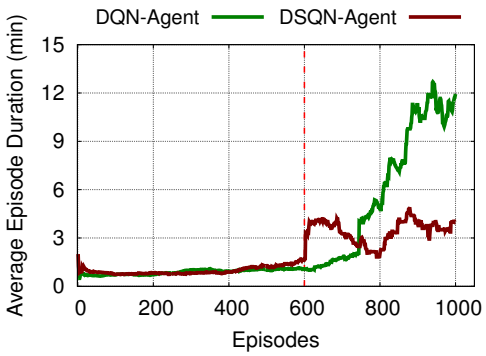


Figure 4.10: Average duration per episode during training.

Figure 4.11: Agents training evaluation: DQN vs DSQN.

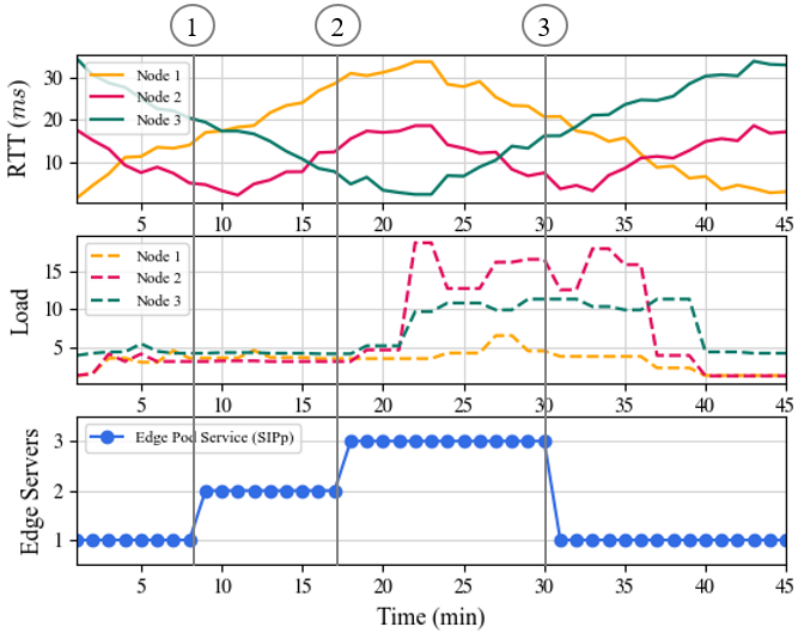


Figure 4.12: DQN agent's actions during user's movement in the highway, in an overloaded edge cluster; vertical lines denote when the migrations take place on pods.

4.5 Conclusion

In this chapter, we developed and experimentally evaluated an SLA-aware 5G edge infrastructure that offers high QoS to the end users regardless of their mobility. We developed the necessary migration capabilities in a k8s environment supporting VM and pod technologies. Our system provides continuous low-latency access to the edge services and an uninterrupted throughput experience. On top of this setup, we implemented a digital-twin environment that is identical to the real-world environment and we developed DQN and DSQN,

migration agents. After the training of our agents in the simulation environment, we employed the DQN agent's model weights in our real-world infrastructure. Our results denote that the DQN agent successfully learned the policy based on multi-cell RTT measurements and the workload of edge servers. Our framework can dynamically and proactively relocate MEC apps in a k8s environment depending on the experience of the users and the condition of the edge nodes. In the future, we foresee extending our scheme to support proactive handover decisions synergistically with service migrations.

Chapter 5

Service Aware Network Slicing for 6G Networks

5.1 Introduction

Edge Intelligence is widely considered the key element for empowering innovation and enabling the beyond 5G and future 6G networks to meet their full potential. It is expected that within 6G, edge intelligence will enable networks to achieve massive performance gains through unique functions and services that take advantage of the close proximity to the Radio Access Network (RAN), while re-program the network operation through the available APIs (e.g. O-RAN for the RAN). Artificial Intelligence is thus playing a major role in this context, allowing the transformation from network observations to key decisions that affect the overall system performance and reliability, even under high traffic loads. [86] Such decisions are fortified through the Multi-access Edge Computing (MEC) architecture, enabling low-latency applications to be hosted over the network with traffic breaking out from the edge to any Data Network (DN) [87].

The cornerstone for all these innovations is the wide softwarization that has taken place in 5G and beyond networks; services that up to the 4th generation were running as monolithic components, locked in vendor-specific hardware, are currently able to be hosted over generic hardware, running as software network functions. The components have been further disaggregated, by specifying standardized interfaces for their intercommunication, realizing a full Service Based Architecture (SBA), capable of instantiating in a cloud-native manner. This approach extends even for the cases of the RAN, for the higher level functions of the base

stations, that can be realized through software functions placed on the edge/cloud, communicating with the Radio Units through high capacity fronthaul links (Cloud-RAN) [88]. The combination of all these features, empowered by Edge Intelligence, creates fertile ground for introducing novel services that manage the virtualized cellular network even in real-time/near-real-time.

Network slicing is a fundamental concept in 5G networks. It refers to the process of creating multiple virtual networks on top of a shared physical infrastructure. Each ”slice” is tailored to meet the specific requirements of a particular service or application, ensuring optimal performance and resource utilization. Although such innovations allow the efficient provisioning of network service under one/more slices with guarantees, usually it is up to the hosted applications to self-adapt to the fluctuations of the network service. For example, in the case of adaptive video streaming, protocols like DASH [89] might request the specific content that can be served over the network, based on the application perception of the network settings (e.g. capacity, jitter, delay, etc.). The disaggregation of network functions, as it has been standardized for 5G, enables the development of further key *xApps* that can take advantage of the APIs, allowing the network to self-adapt based on the applications that are hosted over the top, through the decisions for allocation in the network. Such decisions are usually based on the spectrum allocation (e.g. for Dynamic Spectrum Management [90]), or slicing allocation. In this work, we deal with the slicing part of the network, for automating the slice allocation of the network, based on the services that run on top, thus creating a fully-fledged *service-aware* network.

The development of such functionalities relies heavily on resource disaggregation as defined for 5G networks. This disaggregation has been standardized for different parts of the network (Control/Data Plane and RAN/Core Network) as follows: 1) RAN disaggregation for the base station stack, based on the eight different 3GPP defined functional splits [91], and 2) control and user-plane disaggregation, either at the Core Network side through the adoption of SBA, or the RAN, through the adoption of architectures like O-RAN. In the O-RAN architecture, applications hosted on top at the edge of the network (*xApps* [92]) can retrieve statistics of the base station stack through standardized interfaces and analyze them for inferring features like network load, energy consumption, etc. Based on this inference, they can enforce policies regarding slice allocation and scheduling to ensure the smooth operation of the network. The inference relies on ML models, that can predict the future evolu-

tion of the monitored features/parameters, and thus apply pro-actively the target allocations. The O-RAN architecture can be further enhanced with the Network Data Analytics Function (NWDAF) which is standardized by 3GPP. NWDAF is a network-aware function that collects data from the 5G core and provides statistics to support network automation. These statistics can be employed by AI/ML models that run on RAN Intelligent Controllers (RIC) and can provide forecasting and optimization of Key Performance Indicators (KPIs) [93].

Leveraging Edge Intelligence, ML operations can be launched directly on the edge by taking advantage of several devices if needed in an entirely distributed manner, making use of pipelines. In this work, we design, develop, deploy and experimentally evaluate a service-aware network model for beyond 5G networks. We use a cloud-native network, with the entire stack (RAN and Core Network) being instantiated through the Kubernetes framework. We develop all the necessary extensions to support near-real-time ($\leq 10ms$) low-level monitoring of the traffic exchanged over the network. On top, and towards enabling accurate decisions for the slice allocations in the network, we use a distributed ML model, able to classify in real-time the traffic exchanged from the different users of the network and infer the future connectivity needs that are needed from the applications. The needs are in turn transformed into slice-allocation decisions for the 5G network. Our ML models have been developed in a distributed lightweight manner, allowing different parts of the training process to be executed at/near the edge devices, where processing power is usually limited. By decomposing the main model into lighter components and making extended use of pipelines, we are able to instantiate the framework at the edge and affect the wireless network allocations directly from there, thus augmenting the network with edge-located Intelligence.

Our contributions are summarized as follows:

- To develop a real-time classification model, hosted on the operator side of the network, recognizing the different applications that run on top of the network.
- To infer the future load and patterns of traffic from the different traffic flows of the applications that are hosted on top of the network.
- To decide on the slice allocation that is enforced in the network, based on the foreseen needs of the applications.
- To determine the optimal approach for predicting the future demand, from a set of different supervised ML models.

- To evaluate the developed scheme under real-world settings, using real devices and realistic traffic scenarios in real-time.

The rest of the work is organized as follows. Section 5.2 presents our motivation, based on a recent literature review. Section 5.3 presents our overall system architecture, detailing the different components and their intercommunication, as well as an evaluation of the different ML models that drive our final choices. In Section 5.4.2 we evaluate our contributions and present our findings. Finally, in Section 5.6 we conclude the work and present some future directions.

5.2 Related Work

The disaggregation of the telecommunications stack has been identified as one of the key enablers for flexibility, and further innovations for the beyond 5G and future 6G networks. By taking advantage of the disaggregation and existing approaches for an end-to-end SBA, the telecom stack can be instantiated as cloud-native functions throughout the resource continuum, thus allowing network operators to take advantage/extend existing approaches for VNF management, tailored to network-specific characteristics. Several of the works in the relevant literature focus on managing the deployed components as VNFs, divided mainly into the following categories: 1) Placement of the VNFs [94], [95], [96], [97], 2) load that they are receiving [98], [99], [100], and 3) scale of the functions [101], [102], [103].

The most outstanding effort reflecting these architectural approaches is the definition of the Open-RAN (O-RAN) specifications [92]. O-RAN standardizes the interfaces for interacting in real-time, near real-time, and non-real-time with different components of the RAN stack, enabling the network to re-configure dynamically, based on operator-defined policies. Opening up the programmability of the RAN has created several opportunities for the integration of Artificial Intelligence methods, which infer based on historical observations of metrics on the future resource usage, and appropriately manage the network services.

In the realm of RICs for telecommunication networks, several solutions, both open-source and proprietary, are available. FlexRAN [39] stands out as a flexible and programmable platform tailored for Software-Defined Radio Access Networks (SD-RAN) and is compatible with the open-source OAI platform. Its successor, FlexRIC [40], serves as a software development kit (SDK) designed for next-generation SD-RANs, allowing its customization in

the functions that the user needs to perform on the RAN. On the proprietary front, Athena Orchestrator—O-RAN SMO & RIC [104] is an AI-driven platform optimized for energy-saving management in 5G-ORAN compatible private networks. Additionally, FlexSlice [105] introduces an innovative approach, presenting flexible control logic topologies—centralized, decentralized, and distributed—to refine the O-RAN architecture for reduced control loop latency.

Different methods of Machine Learning are employed for the prediction of different network metrics, depending on the metrics themselves and their fluctuation to incoming load. For example, in [106], authors present a conceptual model for 6G networks and show the use and role of ML techniques in each layer of the model. Different ML methods are examined for the different parts of the stack, including supervised and unsupervised learning and Reinforcement Learning (RL). Regarding supervised learning, they employed Deep Learning (DL) in a distributed manner with the use of Federated Learning (FL). The application of ML has opted in several works dealing with the characterization of traffic exchanged over the network. For instance, in [107], the authors classified the traffic according to application and bandwidth-related features. Furthermore, the networking systems can identify factors that affect the operation of the network (e.g. external traffic for DDoS attacks) and appropriately employ the respective mechanisms for reinforcing the operation of the network (e.g. firewall operation, slicing of traffic, etc.). For example, in [108], authors employ a federated ML approach that can be ideally realized in networking switches, towards detecting intrusions in the network by processing packets at the bit level and at line-speed. In [109] authors use a non-parametric approach for traffic classification, which can improve the classification performance effectively by incorporating correlated information into the classification process, using the nearest-neighbour approach. Their approach demonstrates significant performance benefits from both theoretical and empirical perspectives in the literature. Authors in [110] employ cluster analysis for the case of peer-to-peer networks that use dynamic port numbers for the communication between participating nodes. Their presented approach demonstrates how cluster analysis can be used to effectively identify groups of traffic that are similar using only transport layer statistics. Finally, surveys [111, 112, 113, 114] organize the different traffic classification techniques that have emerged in literature for analyzing traffic based on either their headers, or the payload, and whether it is encrypted or not.

Similarly, in [115] authors propose the adoption of ML for orchestrating different tasks

of 5G and beyond networks, such as massive MIMO, heterogeneous network integration and spectrum access, energy harvesting, and others. In [116], authors introduce the concept of xApps, running on top of the O-RAN architecture. These are network management applications, that rely on statistics exposed from the stack at different levels. Based on the decision time, *xApps* can be running in near real-time or non-real-time fashion. In [117] Thantharate et al. propose the ECO6G model, leveraging a Machine Learning approach to forecast traffic load for improved energy efficiency and OPEX savings in B5G networks. This research demonstrates that ECO6G significantly outperforms traditional forecasting methods in energy savings, presenting a vital step towards sustainable and cost-effective network management.

Regarding the type of policies and enforced decisions, several works deal solely with allocating resources for slicing the 5G network. In [118], authors employ Federated Learning as a means of predicting the evolution of each KPI in a per-service manner. Subsequently, they allocate the slices in the network. In [119], similar functionality is suggested, using the FlexRAN controller for reactively enforcing decisions regarding the network operation. Nevertheless, truly online training and decision-making in such systems pose a significant challenge, as model training can consume slice resources. Authors in [120] propose their solution for combating such issues with an online end-to-end network slicing system, able to achieve minimal resource usage while satisfying slices' Slice Level Agreements (SLAs). In [121] the Probabilistic Intra-slice Resource Service Scheduling (PRSS) algorithm is introduced to optimize 5G network resource allocation. Designed in two stages—service throughput estimation via a multinomial probabilistic model and dynamic conditional resource estimation for new services. Its efficiency is demonstrated through analytical and simulation results, showcasing its capability to efficiently manage 5G network resources.

In this chapter, we developed a solution for enhancing the network operation with intelligence, based on the type of services hosted over the top. By employing a service classifier, we were able to determine in *real-time* the type of application running on the top and decide on the allocation of slices over the network in almost real-time. Moreover, our research stands out by implementing a thorough MLOps strategy, contrary to numerous previous studies that deploy deep learning models on fixed datasets, neglecting the emergence of new data patterns and the ongoing management of the model. To clarify our pivotal contributions within Table 5.1, we present a suite of innovative advancements that distinguish our research from

Table 5.1: Comparison of state-of-the-art with our approach.

Works	Approach	Evaluation
[116]	Open RAN for 6G networks focusing on modular traffic steering implementations.	Highlighted modular approach and AI/ML benefits in simulations; model lifecycle not discussed.
[117]	A supervised ML approach for forecasting traffic load to evaluate energy efficiency and OPEX savings in B5G networks.	Centers on model development and validation using real-world 5G data, omitting live deployment details and lifecycle discussions.
[118]	Uses Federated Learning to predict service-oriented KPIs for 5G network slices, addressing privacy and scalability challenges.	Proven in simulations to enhance KPI accuracy, ensure privacy, and cut communication costs. Highlights gaps in model lifecycle and scalability discussions.
[119]	A RAN runtime slicing system for flexible reactive slice customization in 5G networks, utilizing a runtime SDK for agile control application development.	Prototype development demonstrated on OpenAirInterface and Mosaic5G platforms, focusing on system capabilities.
[120]	Online DRL for dynamic end-to-end network slicing, focusing on SLA satisfaction and resource optimization.	Surpassed rule-based and DRL methods in resource efficiency and SLA compliance in simulations. Omits new traffic adaptation, real-world validation, and lifecycle management.
[121]	Introduces PRSS for optimizing 5G network slicing with a two-stage probabilistic model for resource estimation.	Demonstrated efficiency through analytical and simulation results. Lacks details on deployment, handling new traffic patterns, and model lifecycle management.
[122]	Slices resource orchestration using ML techniques for dynamic slicing of PRBs, admission control, and resource management.	Showcased better prediction and efficiency in simulations against static and random slicing. Lacks real-world deployment details and model lifecycle.
This work	A fully cloud-native, service-aware real-time network slicing model leveraging ML for traffic classification, mobility forecasting, and utilizes MLOps for model lifecycle management with online and distributed training.	Validated in a real-world environment; showcased superior latency and throughput improvements. Emphasizes practical deployment with a focus on adaptability and continuous optimization through a robust MLOps framework.

existing state-of-the-art solutions:

- Leveraged the OpenAirInterface for the RAN and Core Network, running in a cloud-native disaggregated manner using micro-services.
- Utilized programmable attenuators connected to the RAN to simulate realistic mobility scenarios.
- Implemented a custom NWDAF, enriching the dataset with metrics (throughput, jitter, CQIs) for enhanced traffic analytics and mobility insights.
- Used supervised learning to forecast various features and evaluated the solution with 6 different neural networks.
- Introduced and evaluated an MLOps architecture that leverages cloud/edge computing in the resource continuum for Online and Distributed Training among cluster nodes.
- Evaluated the framework in a real-world setup with commercial UEs connecting to the network, generating realistic traffic patterns.

5.3 System Architecture

Our experimental setup consists of a cloud-native disaggregated 5G network fully deployed on the Kubernetes framework. This way, we take advantage of the multiple benefits provided by an application container orchestrator like Kubernetes, such as the management and monitoring of resources and dynamic scaling of the 5G VNFs. The 5G network is enriched by a novel distributed AI/ML unit for continuous distributed training-prediction and slicing. Fig. 5.1 summarizes the framework's architecture, showing the deployment of the service-based 5G network, the introduced distributed AI/ML unit, and the internet applications that the end-users interact with. We deploy the framework in the NITOS testbed [37], a remotely accessible facility located at the University of Thessaly, Greece. NITOS testbed provides Software Defined Radios (SDRs), User Equipment (UE) terminals, and programmable attenuators. All these devices are utilized to develop our solution in a real-world environment. Below, we list the essential elements of our AI network slicing solution that enables provisioning high QoS and continuously user-perceived high QoE.

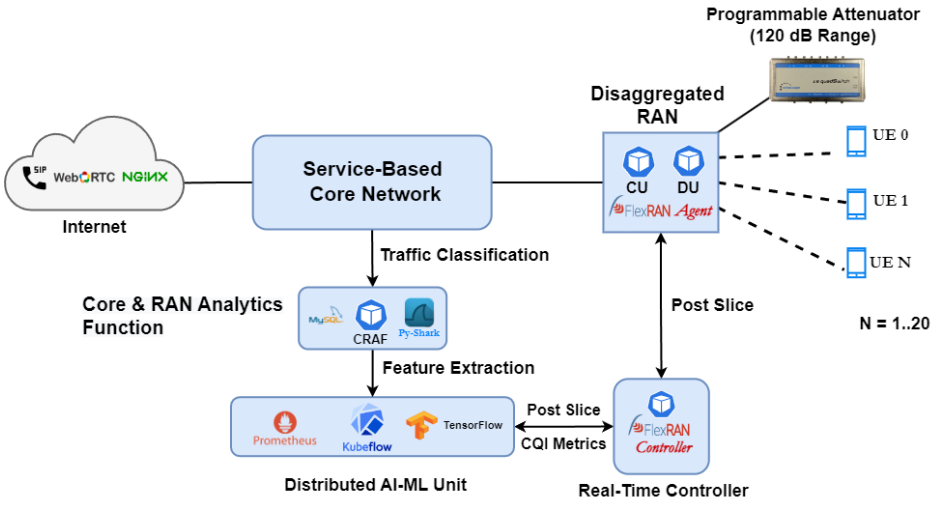


Figure 5.1: Experimental Setup - The deployment of Cloud Native-AI 5G Network on Kubernetes.

5.3.1 Management and deployment of the network functions

Our telecom network follows a serviced-based architecture which consists of containerized network functions. The containerized deployment relies on the open-source OpenAir-Interface platform. We specifically leverage the LTE implementation of the OAI platform, opting for its stability and mature RAN slicing support for multiple User Equipment (UEs), a feature not yet fully developed in the current OAI 5G NR implementation. Despite this, our solution seamlessly integrates with 5G architecture, requiring minimal adjustments to the overall framework. For instance, substituting the LTE Evolved Packet Core (EPC) with 5G core network components (HSS/UDM, MME/AMF, SPGW-U/UPF, SPGW-C/SMF) and transitioning from a disaggregated eNB to a disaggregated gNB can be achieved effortlessly. It's worth noting that our approach to the LTE Evolved Packet Core (EPC) involves the use of Control and User-Plane Separation (CUPS), allowing each component to operate in isolation. Our work focuses on RAN-level allocations, utilizing interfaces envisaged for 6G network operation, such as the O-RAN E2. Notably, our solution remains independent of dedicated slicing components from the 5G architecture, like the Network Slice Selection Function (NSSF). The key distinction with the 5G RAN lies in the absence of full slicing support, with the primary difference being the data rate rather than core functionalities. For the experimental evaluation of our architecture, we created a cluster of three NITOS nodes as Kubernetes workers, while the control-plane node was running on a separate VM. Below, we analyze our cloud-native approach for the deployment of the network functions down from

the core network, up to the end-user.

Service-Based Core Network

The core network architecture follows control and user-plane separation (CUPS). Consequently, each function runs as a separate pod/container providing: a Cassandra database that holds the subscriptions, the Home Subscriber Service (HSS), the Mobility Management Entity (MME), the control plane Service/PDN Gateway (SPGW-C), and the respective user plane service (SPGW-U). Since there's not yet an open-source implementation of the NWDAF we developed a customized function named Core RAN Analytics Function (CRAF). CRAF plays the same role as NWDAF in our architecture. It collects traffic statistics from application interactions and KPI network metrics such as Throughput, Jitter, and the CQI. After the collection of the data, CRAF stores them in a database. Then, our AI/ML framework performs feature extraction and preprocesses the data for the model training.

The fact that the individual core network components run separately as micro-services allows us to easily monitor their status and their consumption in terms of memory, CPU, and bandwidth. The deployment of the core network is distributed to all Kubernetes workers ensuring the load balancing between them. The connectivity between the containerized core network and the Radio Access Network is realized by the Multus Container Network Interface (CNI). Multus CNI allows us to provide multiple interfaces to pods and create static network configurations for easy reproducibility of the experiments.

Disaggregated RAN

The containerized Radio Access Network (RAN) follows a disaggregated architecture including the CU and DU (Central & Distributed Unit) components. This distributed scheme implements the functional split of the base station. Specifically, the split takes place in the layer 2 OSI stack, between Packet Data Convergence Protocol (PDCP) and Radio Link Control (RLC) layers. The CU integrates the upper layers, while the DU integrates the lower layers (from the RLC and below). The communication between CU and DU is based on the F1 Application via the F1 interface. The CU container can be deployed in any of the Kubernetes nodes from our cluster, contrary to the DU pod that needs to be deployed on a specific node equipped with the appropriate SDR front device. In the SDR device, a programmable attenuator is connected, with which we attenuate the signal of the RF device, in order to create

realistic mobility scenarios.

To obtain RAN statistics such as CQI and to create network slices on demand, we utilize the FlexRAN network controller. FlexRAN provides flexible and efficient resource allocation and by this time of writing, is the most stable open-source solution for RAN slicing. We connect the FlexRAN controller to the RAN via the FlexRAN agent running on the CU/DU side. FlexRAN is also connected to the CRAF and AI/ML unit ambiguously for the transmission of the RAN statistics and to the establishment of the slicing policies.

End-Users & Internet Applications

To evaluate the network connectivity and collect traffic data, we connected 3 UEs to the network interacting with 3 containerized applications on the internet. The mobile equipment includes commercial UEs by utilizing LTE dongles. The applications include a video streaming service, a VoIP application, and an Nginx web server. The reason for choosing these services is to classify their network needs into data-hungry applications such as video streaming, medium data-rate applications such as VoIP, and low data-rate applications such as simple web-server. The video streaming service streams video capture devices by utilizing the webRTC protocol as it provides real-time communication over the web. The VoIP service is an application called SiPp that employs Session Initiation Protocol (SIP) for VoIP packet transferring. The Nginx web server is employed for the generation of HTTP requests. All services are containerized and deployed onto the same Kubernetes cluster. This allows us, to deploy them among the SPGW pods on the Node with the SDR device to provide an edge computing approach. Finally, the traffic can be captured and fed to the CRAF, directly from the SGi interface of the data-plane network.

5.3.2 Application-aware AI/ML Unit

Developing an efficient AI/ML unit, aware of the network conditions that coordinates the resources optimally requires considering a lot of parameters. Our approach captures a large number of features, essential for the slicing decision, including the applications used by every UE, the Throughput, and the Channel Quality, among many others. Noticeably, the model receives an input window of multiple time slots, with these features, which represent the network traffic exchanges between the UEs and the applications in the near past. Thus, the model identifies the pattern in the traffic and predicts future values. Our goal is to develop

a robust unit that thoroughly analyzes the overall network conditions and employs a superior slicing allocation algorithm, leading to peak network performance. Below, we provide information on the whole procedure of choosing the proper features, designing an effective traffic classification scheme, creating real-world network traffic scenarios in the experimental environment, collecting data, training multiple models, and developing a novel near real-time slicing allocation scheme.

Feature Selection

Designing a powerful AI/ML model, aware of the plethora of components in a network architecture requires a cautious feature selection. Thus, we pick many features to capture the largest possible variance that explains the pattern underlying the traffic exchanges between the UEs and the applications (apps). Precisely, our features' list consists of the *Applications*, the *Throughput*, the *CQI*, the *Jitter* and the allocated *Slices*, for every UE of the network. First, the *Application/Service* is a principal component of a service-aware implementation capturing which specific service is used by every UE. This feature indicates the service's type, demand, and significance. Importantly, for every UE, we keep one feature for every application provided by the network; in our case, there are 3 app-features (*WebRTC*, *SIPp* and *Nginx*). Further, the experienced service *Throughput* provides essential information about the bandwidth of the UE-App link. Another vital feature is the *CQI* that represents the LTE channel quality, which demonstrates the quality of the UE connection; indicating a great or poor connection. Moreover, the *Jitter* monitoring per UE depicts the timing delays between the UE's packets, while the *Slices* show the allocated resource blocks of every UE.

Traffic Classification

For traffic classification, we divide the timeline of every experiment into multiple time slots of a fixed length, in which we gather the desired network information with the aforementioned features.Importantly, the information in every time slot is organized in a specific structure. We divide every time slot into multiple UE categories as shown in Fig. 5.2. This way, the information for every UE is gathered in one category. In our case, there are 6 different features for every UE category, namely *WebRTC, Sipp, Nginx, CQI, Jitter* and *Slice*. The first three features represent the network *Services* that the UE is able to use. Noticeably, their values represent the *Throughput* of the specific UE with the specific service. For in-

stance, a value of 10 in the *WebRTC* feature in the first category (UE 1) is translated as 10 Mbps network traffic on the UE 1 using the *WebRTC* service. The remaining features of every category, namely *CQI, Jitter* and *Slice* provide additional information on the quality of the UE connection as well as its allocated resources. As a result, we end up with a number of columns that is proportional to the number of UEs multiplied by the number of features per category; in our case, 3 UEs multiplied by 6 features equals 18 total columns (real features for the model) for every time slot. This is illustrated in Fig. 5.2, where every column of the tables is a feature and every row is a time slot.

This way, we organize the monitored network traffic into a useful structure to be used by a model. Precisely, the time slot length is configured to the desired number, for instance, 100 ms. Subsequently, during every slot, we gather all the received packets and extract the essential information. Firstly, we read the packets' IP/Transport protocols to classify them to the appropriate UE-App combination. Then, we count the total number of bytes of all packets received during the time slot for every UE-App link to calculate the Throughput. This way, we classify the captured traffic during a time slot to the appropriate columns. Next, we compute the mean Jitter value between the total packets of every UE in the time slot. On top of that, a CQI value per UE is requested from the FlexRAN Agent existing in the LTE DU, and finally, the currently allocated UE slices are recorded as well. For a better understanding, let's focus on Fig. 5.2 in the first row of the third input window ($i = 2$). The first 6 values corresponding to the category of the UE 1 are:

$$(A, B, C, D, E, F) = (10, 0, 0, 14, 1, 8)$$

Interpreting this category, we understand that the UE 1 has 10 Mbps network traffic only with the WebRTC service, an LTE CQI of 14, 1 ms average Jitter, and allocates a slice of only 8% of the overall network resources.

Real-world Traffic Scenarios

We emulate realistic network behavior in an office by developing multiple network traffic scenarios. Our goal is to emulate inside our experimental infrastructure the network patterns observed in an office on a specific time interval of a usual day. We aim at specific time intervals and not the whole day since our resources are limited. Most users in an office are expected to have a basic pattern in their behavior. For example, one user might mainly utilize

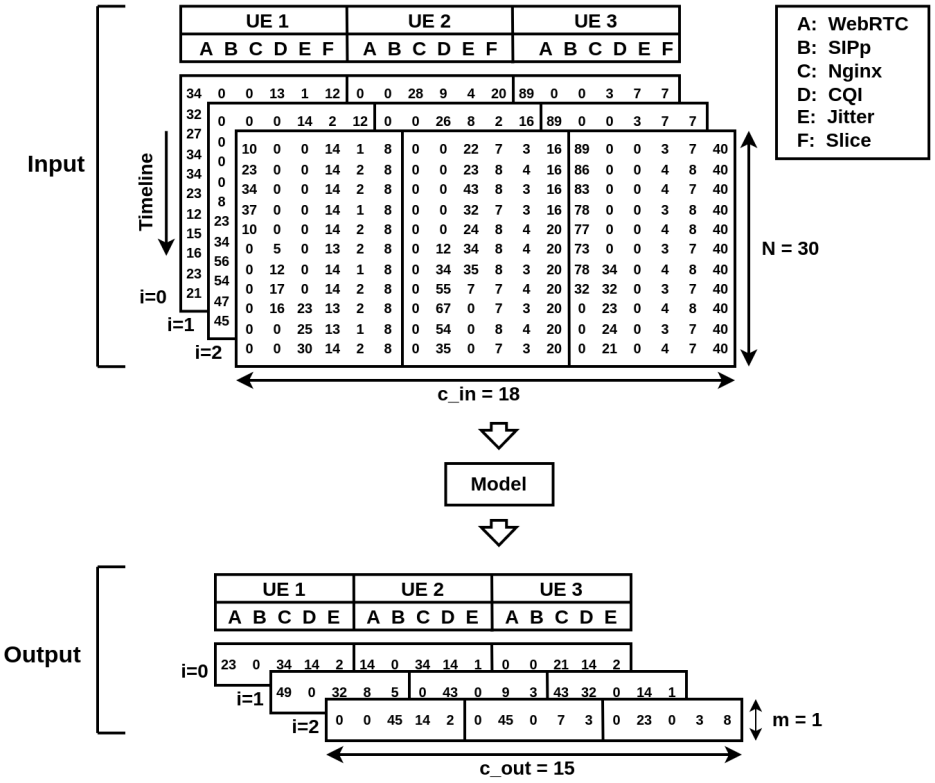


Figure 5.2: Traffic Classification & Sliding Window Approach

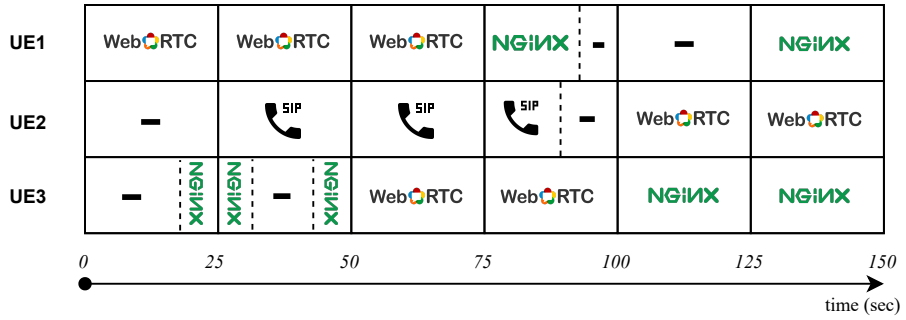


Figure 5.3: Users' Network Traffic Baseline Scenarios depicting network traffic at a specific time interval during the day.

video streaming platforms, whereas another one is constantly on calls with clients. Thus, the AI/ML unit captures this pattern and enhances users' overall experience by sharing the network resources on demand. As a first step towards emulating this office behavior, we create some baseline traffic scenarios for every UE in our network (one bash script per UE specifying a particular behavior) as shown in Fig. 5.3. These scenarios are based on real network patterns observed at a specific time interval during the day (early morning from 10:00 AM to 11:00 AM) on users in our office facilities in Volos, Greece. However, we

redesign them to be small with a duration of approx. 150-160 seconds to facilitate the whole experimental procedure on the testbed. This way, we create the basic pattern that is observed in our office at that specific time interval. However, this is not the exact behavior every day since it will slightly change from one day to the other even if the underlying pattern is the same. For example, the employee who works mainly on the phone will not make the same number of calls or calls of the same duration every day, but he/she will mainly work on the phone with clients. To emulate these slight variations in the UE behaviors from day to day, we employ data augmentation techniques. Specifically, based on the baseline scenarios, we add Additive White Gaussian Noise (AWGN) in the number, sequence, starting time, and duration of the utilized applications by a UE to represent the differences from one day to the other. For instance, the UE 3 in Fig 5.3 uses the WebRTC app one time starting at 50 secs for a duration of 50 secs. It also uses the NGINX app three times in total each starting at about 20, 45, and 100 secs for a duration of 10, 5, and 50 secs respectively. At first, AWGN from the standard normal distribution with a mean of 0 and a standard deviation (sd) of 1 is added to the number of times that an app is used. Regarding UE 3, this means that the number of times that the WebRTC and NGINX are utilized will either not change or increase/decrease up to a maximum of 3 times (3 standard deviations from the mean). Then according to the new numbers we add the new apps or delete the unnecessary ones randomly. Subsequently, we use the same distribution to choose randomly several apps (up to three) and change their position in the timeline. Then, AWGN from a different distribution (mean of 0, sd of 10) is added to change the starting time of each app up to a maximum of 30 secs (3 sd from mean). After that, AWGN from the same distribution is inserted to change the duration of each app increasing or decreasing it by a margin (up to 30 secs - 3 sd from mean). At every step, we adjust accordingly the position of the apps in order to avoid interference.

Moreover, several scenarios are reversed to augment the dataset further and a lot of them are slightly cropped for efficient training. Further but minor noise is inserted when we collect the data from the testbed due to hardware imperfections. Thus, we create a plethora of network traffic scenarios for every UE that inherit the baseline pattern but are slightly modified capturing a large spectrum of the office's real traffic at that specific time interval. Hence, there is a large variance to build robust AI methods, capable of generalizing, not over-fitting, and being resilient to noise and fluctuations.

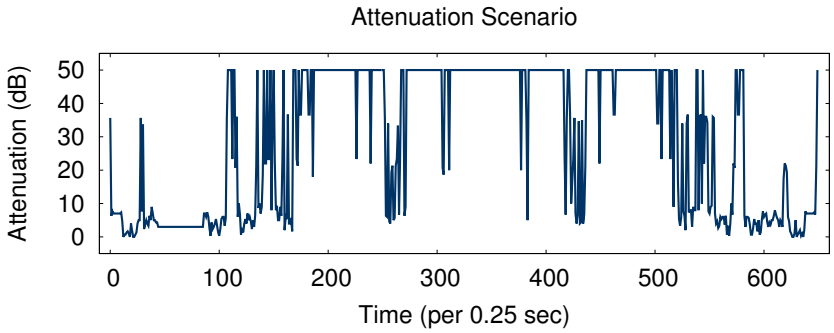


Figure 5.4: Attenuation Scenario emulating UE mobility in office.

UE Mobility Emulation

In a real network, the quality of the UE connection varies according to the geographical location of the UE. Specifically, in areas with good LTE coverage the CQI that depicts the LTE channel quality, is high, in contrast with areas where there is poor LTE coverage (low CQI). In order to emulate this behavior in our experiment we use programmable attenuators installed on the outputs of the USRP, as presented in Fig. 5.1. Specifically, by modifying the attenuation of the USRP radios, we can emulate transitions from low to high CQI values and vice versa. The attenuation is inversely proportional to the CQI (high attenuation causes low CQI and the opposite). Importantly, we possess attenuation scenarios from real commercial networks in Volos, Greece. Specifically, these attenuation scenarios emulate cars traveling a specific city route with velocities that vary from 40 to 60 km/h with the road's limit being 50 km/h. These car scenarios were used to collect 182500 CQI data from 73 cars capturing a large spectrum of the route's traffic. The CQI data are publicly available [123]. We decide to utilize the same attenuation scenarios to emulate mobility to the office users since it is a similar problem (users moving in a specific geographical area) and moreover, because it is a dataset with a large variance that could lead to efficient training and generalization of the models. Fig. 5.4 depicts an attenuation pattern used, where at the beginning of the experiment the attenuation is low (high CQI). Following that, the attenuation rises substantially (low CQI), while at the end of the experiment, the attenuation returns to low levels (high CQI).

Data Collection

To collect a lot of training examples for our model, we execute all the scenarios in the testbed. In specific, we pick at random one of the traffic scenarios (office users' pattern) and

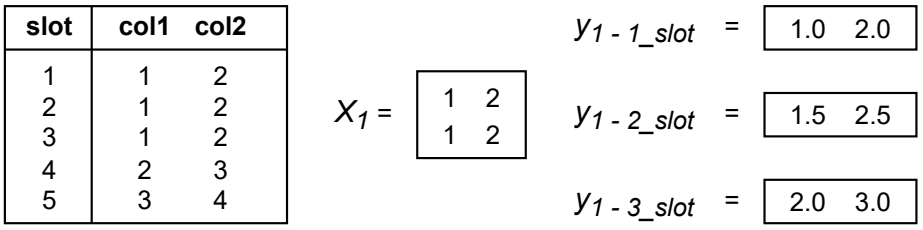


Figure 5.5: Example of sliding-window scheme.

one of the attenuation scenarios (mobility pattern) and execute them concurrently. This way, we assign a different combination of office traffic and mobility patterns to each experiment. Meanwhile, by employing the traffic classification scheme with a time-slot duration of 1 second, the network traffic is appropriately classified and subsequently stored in the database. This is done for 300 experiments (each lasts approx 150-160 seconds) creating, as a result, a massive dataset with 48600 rows and 18 columns. This dataset is also publicly available [124].

Pre-processing

Before feeding the data into the models, we need to preprocess them appropriately. First, we normalize the whole dataset adjusting all the columns in one common range between 0 and 1. This way, we avoid scale imbalances strengthening the model's training efficiency. Subsequently, Fig. 5.2 illustrates clearly our pre-processing technique. In specific, we utilize a sliding-window approach which creates a 2D input window (X_i) of fixed shape ([N time slots, c_{in} features]) and slides it by one-time slot over the whole dataset to create multiple samples ($i = 0, i = 1, i = 2$). Meanwhile, for every X_i sample, the algorithm captures a second 1D output window (y_i) with shape [1, c_{out} features], which depicts the data that we want to predict (labels) The data of every 1D window (y_i) are located immediately after that of the 2D window (X_i) in the dataset representing the future. Noticeably, the values of each y_i could be that of only one-time slot (the following of the X_i) or the mean values of an arbitrary number of time slots following the X_i. For example, we provide a dataset with shape [5,2] in Fig. 5.5

Given that we want to pick X_i windows with a length of 2-time slots, the first input sample (X_1) would be the first two rows. For the corresponding prediction-output window y_i there are a lot of choices depending on the number of future time slots that we want to predict. For instance, to predict one future time slot, the y_1 would be the third row. On the other side,

to predict multiple future time slots, one efficient solution is to obtain the average values of their columns. Fig. 5.5 demonstrates examples for predictions of 1, 2, and 3 future time slots:

For the following X_i, y_i samples, we slide by one-time slot and apply the same procedure until we reach the end of the dataset. In our case, as shown in Fig. 5.2, after extensive experimentation we conclude on calculating X_i windows with shape [30,18] and y_i vectors of shape [1,15] predicting the average values of five future time slots. The general rule for finding the optimal window shapes is that the X_i windows should be sufficiently large to capture the pattern in the near past but small enough to boost model training and avoid the exploding/vanishing gradient problem when Recurrent Neural Networks (RNNs) are used. Regarding the number of future time slots for prediction, it is generally good to employ multiple future time-slots to smooth possible fluctuations, but not too many of them so as to present an accurate figure of the near future. Using this technique, we structure the data in X_i samples of shape $[48566, 30, 18]$ and y_i samples of shape $[48566, 15]$.

Neural Network Models

This work focuses on supervised learning approaches and specifically, on evaluating various deep learning methods. We focus on neural networks as they are generally more robust at handling huge datasets and more resilient to noise compared to statistical and tree-based methods.

Our goal is to design a robust Neural Network (NN) that converges on the pattern fast and accurately in order to be used for real-time forecasting implementation. Hence, we explore many different NN structures and finally conclude on some of the most promising ones and provide their specifications in Table 5.2.

Firstly, we choose an FNN due to its simplicity by just moving the information forward from the input to the hidden and to the output layers resulting in faster training. Subsequently, we move to more sophisticated architectures, the RNNs, which employ memory components and are widely utilized in Time Series Forecasting (TSF). Precisely, LSTM NN are very robust at dealing with the vanishing/exploding gradients issue using three gates (input, output, and forget gates) and thus, they often outcompete simpler RNNs. Following that, we extend the simple LSTM by inserting a Bidirectional layer (Bi-LSTMs). This way, the model analyzes both the original sequences and their reversed versions, obtaining information from the past and also the future, usually resulting in enhanced forecasting performance. After that, we

analyze GRUs NNs, another widely used RNN, that achieves similar predictive performance with LSTMs. In fact, GRU is equipped with fewer gates (reset and update gates) and hence, requires fewer training parameters leading to faster training. Then, we build a CNN that is powerful at efficiently extracting features, dealing with noise, reducing the dimensions, and calculating non-linear functions in data by employing kernel filters, pooling layers, and fully-connected layers. Consequently, they often result in more accurate and fast training. Further, we experiment with a hybrid CNN-LSTM that obtains the best from both worlds by forming an Encoder-Decoder architecture. In specific, the CNN part implements feature extraction, noise, and dimensionality reduction and subsequently passes the processed information to the LSTM, which captures the pattern in data using memory components. This way, the result is a prominent model with remarkable predictive and training performance.

Table 5.2: Neural Networks Configuration

Model	Layers	Hidden Layers	Epochs
GRU	2 GRU + Dense	25 units per layer	61
LSTM	2 LSTM + Dense	25 units per layer	97
Bi-LSTM	2 Bi-LSTM + Dense	25 units per layer	56
FNN	2 Dense + Output Dense	25 units per layer	568
CNN	Conv1D + MaxPooling1D + Flatten + Dense + Output Dense	Filters=64, Kernel size=2, Pool size=2, 25 units per Dense layer	264
CNN-LSTM	Conv1D + MaxPooling1D + Flatten + RepeatVector + 2 LSTM + Output Dense	Filters=64, Kernel size=3, Pool size=2, Repeat factor=1, 25 units per LSTM layer	24

Slicing Allocation Mechanism

The slicing allocation algorithm is designed to provide the network resources on demand and fairly to maximize the QoE of the UEs. To achieve that we share the available network resource blocks based on a mathematical formula that consists of many criteria obtained from the model predictions. Precisely, the type of the application (C_1), the total Throughput of the UE (C_2), the CQI (C_3), and the Jitter (C_4):

$$Slice(\%) = \sum_{i=1}^{4} (w_i C_i) + w_0, \tag{5.1}$$

where w_1, w_2, w_3, w_4 are the weights of every criterion indicating its importance and w_0 is a constant term representing the minimum value of the slice.

Each criterion (C_i) is assigned a priority value (0, 1, or 2), signifying low, medium, or high importance, respectively. For example:

- For UE application (C_1), WebRTC is given the highest priority (2), followed by SIPp and Nginx with priorities 1 and 0 correspondingly.
- Throughput (C_2) is classified as high demand (2) for values above 0.4 Mbps, medium demand (1) for values between 0.2 and 0.4 Mbps, and minor demand (0) for values below 0.2 Mbps.
- CQI values (C_3) falling between 0 to 9 are high priority (2), 9 to 11 are medium priority (1), and above 11 are low priority (0).
- Jitter values (C_4) of more than 10 ms are crucial (2), 5 to 10 ms are medium priority (1), and less than 5 ms are low priority (0).

After experimenting with various slice configurations, we determined that in our experimental setup, maintaining a minimum slice value of 8% is crucial to keep a User Equipment (UE) connected to the network. Any value below this threshold results in UE disconnection, prompting us to establish 8% as the designated minimum slice value (w_0). Additionally, we observed that UEs achieve their optimal performance when allocated a slice of 40%. Beyond this value, there is no discernible increase in connection efficiency. Consequently, we selected 40% as the maximum slice value. This maximum value is determined when all criteria in Eq. 5.1 have the highest priority:

$$40 = w_1 \times 2 + w_2 \times 2 + w_3 \times 2 + w_4 \times 2 + 8$$

In our study, we assigned equal importance to each criterion, reflected in identical weight values for w_1, w_2, w_3, w_4, all calculated as 4. Consequently, the slicing equation simplifies to:

$$Slice(\%) = 4 \sum_{i=1}^{4} C_i + 8 \tag{5.2}$$

Various strategies can be implemented by assigning different weights to individual criteria based on specific objectives. For instance, prioritizing Ultra-reliable Low Latency Communications (URLLC) would involve assigning a higher weight to the *Jitter* criterion (C_4). This adjustment enhances the slice allocation sensitivity to Jitter, ensuring that more resources are allocated to UEs experiencing Jitter fluctuations. Alternatively, assigning greater weight to *Throughput* (C_2) could strengthen support for Enhanced Mobile Broadband (eMBB), while an emphasis on the weight of *CQI* (C_3) would focus on maintaining a stable, high-quality connection. Similarly, allocating more weight to *Application* (C_1) would result in additional resources based on the application type rather than the quality of the connection.

In our case, we choose an equal weight to all criteria to evaluate the algorithm's general efficiency as a first step. Future works will focus on specific use cases. Table 5.3 adduces examples of the slicing allocation algorithm for further understanding. For instance, the forecasting regarding the UE 1 indicates that the Nginx app will be utilized with 0.1 Mbps Throughput, a CQI of 14, and a Jitter of 2 ms. All these values correspond to the lowest priority (0) of each criterion (C_i) and thus, the calculated slice is the lowest, 8%. At UE 2 and 3, all criteria have medium and maximum priority leading to a slice of 24% and 40% respectively.

When the total slices of the UEs are calculated more than 100%, we subtract an equal proportion of every slice. Overall, the UE receives the appropriate amount of resources depending on the network conditions without under or over-provisioning. In general, this scheme could be adapted to individual preferences. First, further criteria could be added or some of them could be excluded. Secondly, the weights could be adjusted on the individual preferences to target specific use cases. Additionally, the minimum and maximum values of the UE slice could be modified. Finally, this Eq. is a linear relationship between the criteria and the slice, and thus in the future, it could be replaced by a non-linear function calculated by an ML model.

Forecasting	**UE 1**	**UE 2**	**UE 3**
Application	Nginx	SIPp	WebRTC
Throughput (Mbps)	0.1	0.3	2
CQI	14	10	6
Jitter (ms)	2	8	12
Criterion	**UE 1**	**UE 2**	**UE 3**
C_1	0	1	2
C_2	0	1	2
C_3	0	1	2
C_4	0	1	2
Slice(%)	**8**	**24**	**40**

Table 5.3: Examples of UE slices assigning the priorities to each criterion (C_i) based on forecasting.

5.3.3 MLOps AI-ML Unit Architecture

To ensure that our model adjusts to the training data's gradual drift, we employ an online/distributed training architecture realized by a Kubeflow pipeline. Kubeflow is an open-source AI/ML toolkit that utilizes the power of Kubernetes to run ML jobs and supports the entire lifecycle of ML applications. In Kubeflow, a pipeline is a description of an ML workflow that includes containerized components, each of which represents a single step in the process. Each element is managed as a microservice, with all the expected declarative definitions (YAML manifests). This, enables them to be quickly deployed and scaled out as required. By employing Kubeflow [44] pipelines we can easily orchestrate, scale, and automate our AI solution. This MLOps - Distributed Architecture is presented in Fig. 5.6. First, CRAF monitors all the traffic from the SGi interface by utilizing PyShark [125]. In order for CRAF to collect the traffic in real-time, we use the *LiveCapture* class of PyShark. CRAF also obtains all the CQI values in real-time, via HTTP requests from the FlexRAN controller. Then, after applying network filters to the traffic (IPs/Ports), it classifies the interactions per UE and application and calculates traffic analytics such us Throughput and Jitter. To avoid big data over time, CRAF only keeps the summary of each packet such as the UE, the Application, the Length, the Jitter, and the CQI value that each UE experiences. Subsequently, this data is stored on a database running on a MySQL server that is backed with NFS per-

sistent storage via PersistentVolume, providing consistency and availability of data between Kubernetes Nodes. Next, the Kubeflow pipeline takes place, as the first step: the Data Parser extracts the features from the database and creates a new dataset. Then, the next pipeline component, the Data Preprocessing applies the sliding window approach to the data and stores them in a multi-dimensional array. Afterward, this newly shaped array is passed to the last step of the pipeline, the Training component. The construction, and the training of the model, are implemented in this final step. After the train finishes the new model is saved on the NFS as an HDF5 file via the mounted Persistent Volume that is attached to the container. This way, the Predictor Service can obtain and utilize the updated model as it has access to distributed storage as well. As a result, the Predictor pod can make live predictions for near future traffic with higher accuracy, as the model is trained with the data with the most recent interactions and the latest network conditions.

To calculate the overhead of our solution we rely on the Eq. 5.3. It is the total time that is needed per slice allocation. All the metrics are measured with the help of timeit python module. The first metric, t_{CRAF}, is the total time for CRAF to obtain traffic and RAN analytics in one iteration. We measured that t_{CRAF} is almost real-time: 1-6 ms. The time needed for slice allocation t_{apply} is also in the same real-time range. This seems reasonable since CRAF employs PyShark for live packet capturing and FlexRAN for RAN statistics, which operates in real-time. Also, the overhead of each prediction (t_{pred}) is 1.6 ms. Finally, the catalytic factor of Eq. plays the time slot per X_i observation described by t_{slot}. We choose to observe X_i every 1 sec to get a better picture and capture the patterns. However, the time slot is a hyperparameter that can be changed. The smaller it becomes, the faster the slice allocations, with the only tradeoff being the efficiency of the predictions.

$$Slice_{time} = t_{CRAF} + t_{slot} + t_{pred} + t_{apply} \tag{5.3}$$

The pipeline can be triggered by the Predictor Service periodically with a timer or each time the predicted data is less accurate than a predefined threshold. This can indicate that the new data that is fitted into the model has different traffic patterns than the data that the model has been trained with. In that case, an algorithm 3 is suggested. As long as the accuracy (*R-squared*) of the forecasts is high, the slice decisions defined by the slicing Eq. 5.1 can be determined by the predictions. Otherwise, if the accuracy is lower than the accuracy threshold, then the slice decisions will be reactively determined by the slicing Eq. directly. The tradeoff

in this approach is the fact that in the middle of the train of the updated model, we might lose some important interactions of the users with the applications as well as the new patterns of the network conditions (e.g. low CQI values). However, based on our experiments this algorithm can converge on new traffic patterns over time as the accuracy remains at constant-high percentages from one point onwards.

Algorithm 3: Online Training and Prediction

```
Function model_select_predict():
    train_flag ← 0
    while True do
        traffic_data ← get_traffic_data()
        accuracy ← get_accuracy_of_predictions()
        if accuracy ≥ accuracy_threshold and train_flag == 0 then
            yhat ← predict(traffic_data)
            store_predictions(yhat)
            slice_perc ← slice_decision(yhat)
        else if accuracy < accuracy_threshold and train_flag == 0 then
            slice_perc ← slice_decision(traffic_data)
            trigger_training_pipeline()
            train_flag ← 1
        end
        else if train_flag == 1 then
            slice_perc ← slice_decision(traffic_data)
            if pipeline_status() == complete then
                train_flag ← 0
            end
        end
    end
End Function
```

Towards aiming to reduce training time as much as possible and to distribute the training load evenly in the Kubernetes cluster, we enrich our architecture by employing Distributed

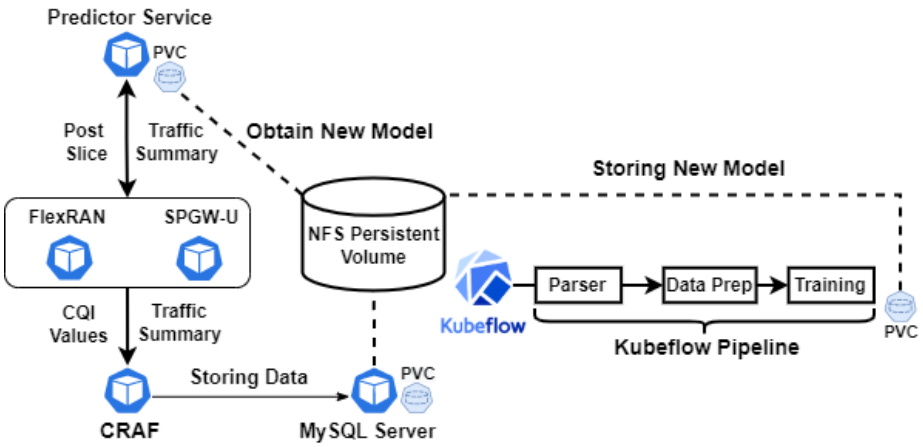


Figure 5.6: MLOps Training Architecture.

training using Kubeflow's TensorFlow operator. With the TensorFlow operator, we can run distributed TensorFlow jobs (TF jobs) in our Kubernetes cluster as illustrated in Fig. 5.7. A distributed TF job is the collection of the following processes:

- Chief: Is responsible for orchestrating the training process
- PS: Parameter Servers provide a distributed data storage for the model parameters and perform gradient updates.
- Worker: The workers do the actual work of training the model.

Kubeflow handles the above processes by passing the Kubernetes cluster configuration as an environment variable to the TF jobs. We only define distributed strategies into our code for synchronous training based on the all-reduce algorithm or for asynchronous training via parameter server. In our experiments, we choose Multi-Worker with All-Reduce strategy and RING communication as it supports synchronous training, without suffering from bottleneck communications, contrary to the parameter server asynchronous training [126]. The distribution scheme can be further extended by describing the training job with a custom YAML file that references the TFJob Custom Resource Definition (CRD). In this way, we can scale our training process into multiple pods that will train the model in a distributed fashion taking advantage of the total resources of the cluster.

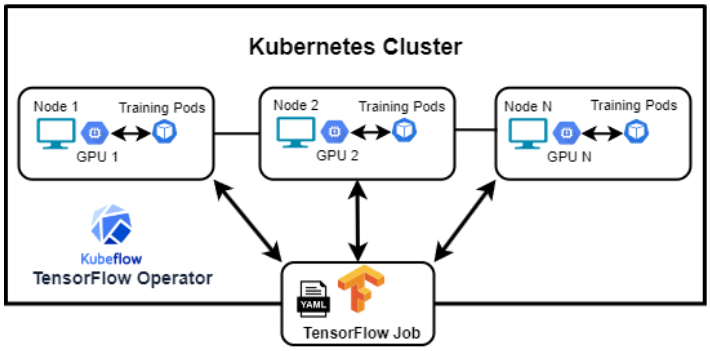


Figure 5.7: Distributed Training

5.4 Evaluation

5.4.1 Model Comparison

The models' offline training and evaluation are taking place on Google Colab where non-subscription TPUs are used. The concluded/optimal model structures are analyzed in Table 5.2. To evaluate them, we employ Time-Series Cross Validation (CV), a technique similar to K-fold CV but designed to respect the time sequence. We split the pre-processed data (48566 X_i, y_i samples) into several folds of equal size (500 samples) and create two sets; the training and the testing one. At first, we initialize the training set with multiple serial folds following the timeline (32000 samples - data of about 200 experiments). On every iteration (i), the model is trained on the training set and uses the next fold on the timeline as a testing set to calculate the generalization error on unseen data. In the following iteration, the training set is increased by one fold following the timeline, and the next one is used for a new evaluation. In the end, the mean of all testing errors (data from about. 100 experiments) is calculated as the overall generalization error. As a second step, we pick each model and integrate it into our experimental topology to evaluate its predictive performance in realistic circumstances on our Testbed. The time-series CV and Testbed's experimental evaluations are shown in Fig. 5.8

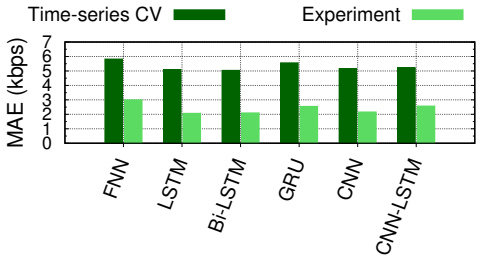
As evaluation metrics, we employ the Mean Absolute Error (*MAE*) and the Coefficient of Determination (R^2). MAE finds the mean absolute error between the predictions ($\hat{y}_i$) and the labels. It is scale-dependent helping us understand the forecasting error when studied together with the data range and distribution. We calculate separate MAE values for the predicted UE-App Throughput, UE Jitter, and UE CQI both for the Time-Series CV and the Testbed's experimental evaluation, as shown in Fig. 5.8. Regarding Throughput, we observe

a range of 0-800 kilobits per second (Kbps) with poor slicing and a range of 0-4 megabits per second (Mbps) with maximum slicing when the utilized application is the WebRTC. On the other hand, when Nginx and SIPp are used, the range is between 0-300 Kbps. Generally, the observed Throughput range in our experiments is between 0-4 Mbps. Regarding Jitter, the observed range is between 0-70 milliseconds (ms) depending on the link quality, slice, and application. Moreover, CQI ranges from 0 to 15. Further, we employ the R^2 metric, which calculates the proportion of total variation of outcomes explained by the model. It is more intuitively informative (percentage value) without the need to consider the data ranges.

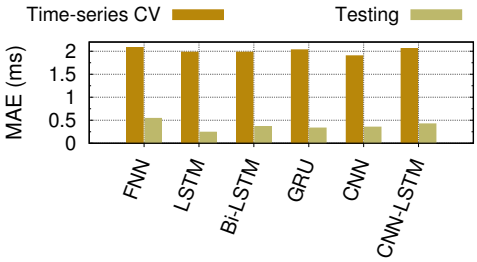
In Fig. 5.8 all the models identify the pattern in data efficiently. In specific, in Fig. 5.8α′ the models have time-series CV Throughput MAE values that range from 5.04 to 5.82 kbps, while the respective ones on the Testbed range from 2.07 to 3 kbps. These error values are negligible when compared with the throughput range, which is 0-4 Mbps. Additionally, the NNs predict accurately the experimental Jitters (Fig. 5.8β′) reaching MAE values at just around 0.25 ms; very minor when studied with the Jitter range of [0-70 ms]. Moreover, regarding the CQI in 5.8γ′, the models achieve exceptionally low testing error with an average of 0.42 MAE considering that CQI ranges from 0 to 15. Moreover, the evaluation utilizing the R^2 metric on the time-series CV and on the experiments on the Testbed are shown in Table 5.4. Overall, the NNs achieve substantial performances, with each model being slightly better in forecasting different features. Importantly, there is a great discrepancy in their training time, as shown in Fig. 5.8δ′. The CNN-LSTM identifies quickly the patterns requiring only 4 minutes, while the remaining models demand from 26 to 76 minutes. The key enabler of CNN-LSTM's training efficiency is its convolutional (CNN) part. In specific, the CNN performs optimally feature extraction, noise, and dimensionality reduction. As a result, the LSTM part finds smaller and better-structured sequences being able to converge on the patterns in a faster way. Thus, we pick this algorithm and integrate it into the AI/ML unit as it combines high predictive accuracy with extremely low training time, being the most appropriate choice for our implementation.

Table 5.4: R^2 Evaluation of the Neural Networks

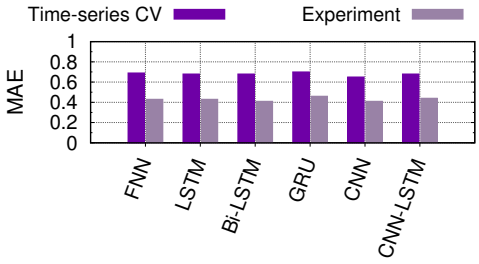
	FNN	**LSTM**	**Bi-LSTM**	**GRU**	**CNN**	**CNN-LSTM**
Time-series CV R^2	0.936	0.940	0.940	0.937	0.945	0.940
Experiment R^2	0.985	0.986	0.987	0.986	0.987	0.986



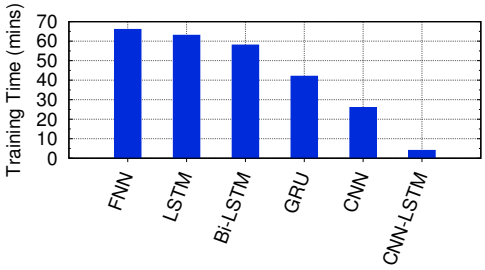
(α′) Throughput MAE per Model



(β′) Jitter MAE per Model



(γ′) CQI MAE per Model



(δ′) Training on Google Colab

Figure 5.8: Model Off-line Training Evaluation on Google Colab and Experimental Evaluation on Testbed.

5.4.2 Experiment Evaluation

Our real-world experiment on NITOS Testbed evaluates the impact of the AI/ML unit using the CNN-LSTM model on the QoE of the UEs. In Fig. 5.9-5.11, we include five different sub-figures for every UE. In specific, the first two subplots (a, b) depict the utilized services, followed by the experienced Jitter (c), then the CQI (d) and subsequently the allocated slices (e) that were provided according to the dynamic slicing allocation algorithm. Noticeably, we compare the resulted QoE of the UEs between the guidance of the AI-unit and the default network configuration. As a ”default” configuration, we set all the UE’s slices to an equal percentage of 8% during the whole experiment. This is done in an effort to show the results of poor resource management by a fair resource allocation algorithm that provides fixed and equal resources to every UE. Noticeably, a slice of 8% is the minimum that keeps a UE connected to the network in our topology. Moreover, it suffices for the light applications, namely the NGINX and SIPp, on maintaining a high-quality connection. However, the most resource-intensive application, WebRTC, suffers from a lack of resources with a slice of that value. On the other side, a fair algorithm that assigns a slice of 33% to every UE (maximum possible by the default configuration) provides enough resources to all apps but leads to a massive over-provisioning. In specific, it wastes huge amounts of resources for the NGINX

and SIPp that could be used to enhance the QoE of the other UEs. Thus, our target is to utilize the AI unit to dynamically and efficiently allocate the slices avoiding over-provisioning to NGINX and SIPp and under-provisioning to the WebRTC.

To begin, the subplots 5.9α′ and 5.9β′ illustrate the apps used by UE 1. At first, UE 1 interacts with WebRTC until approx. 80 secs, when the Nginx is used in two bursts (70-90 and 120-150 secs). Moreover, the Jitter experienced with the default network slicing is higher initially and gradually decreases, while the CQI fluctuates around low values (6-10) almost during the whole experiment. Noticeably, the algorithm provides the slices on demand by increasing the resource blocks at the maximum of 40% in the first part of the experiment (until 80 seconds), where the demand is clearly higher; the UE interacts with the WebRTC, the Jitter is high and the CQI is poor. Subsequently, the demand declines as the UE 1 switches to the Nginx, the Jitter values decrease and the CQI rises until it plateaus to around 13, at the end of the experiment. Therefore, the slicing percentage gradually decreases until it plunges at the minimum of 8%, after around 140 seconds. This slicing management contributes positively to the QoE of UE 1. Specifically, by comparing the network performance of the default slicing algorithm with the AI-unit's, we can see that the Throughput increased reaching even 1 Mbps with the AI-unit when it used to have around 0.2 Mbps as shown in Fig. 5.9α′. In Fig. 5.9β′, we do not observe any changes since the Nginx is not demanding and it has already reached its peak with the default slicing. Finally, the Jitter falls to lower levels at approx. 10 ms with the guidance of the AI-unit from the 30 ms that it used to be.

Regarding UE 2 (Fig. 5.10), we see the opposite behavior. In particular, at first UE 2 interacts with the SIPp until approx. 100 seconds, when it switches to the WebRTC. Moreover, the Jitter remains constant during the whole experiment at 15 ms, as shown in Fig. 5.10γ′, while the CQI seems to slightly fall at 10-12 values (5.10δ′). Noticeably, the algorithms provide a low percentage of resource block at first until around 80 secs, where the demand is relatively low since the quality of the connection is quite good (CQI and Jitter) and the utilized service, the SIPp, is of medium priority. Later, the provided resources are moderately increased to an average of 28% due to the usage of the WebRTC. Importantly, they do not reach higher levels as the link quality is still quite good. Consequently, the QoE of the UE 2 is substantially peaked. Particularly, the WebRTC reaches 4 Mbps Throughput with the AI-unit when it used to reach only a negligible amount of 0.5 Mbps with the default configuration.

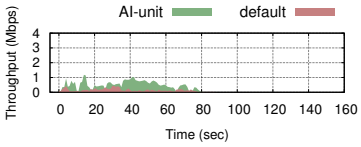
Regarding UE 3, the WebRTC is used in the middle part of the experiment, from 50 to

approx. 100 secs. Additionally, the Nginx is used majorly in the second part at around 100 secs. The Jitter values are relatively low at an average of 5 ms given that the CQI is extremely high (15) almost during the whole experiment, except for the last 30 secs when it slightly declines to around 11. For these reasons, we observe that the slicing allocation mechanism provides few resources during the first part (no more than 16%) until approx. 70 secs, when the WebRTC Throughput is substantially increased demanding more resources. Then, the algorithm raises the resources to 28% and subsequently drops them to 12% at around 100 secs since the WebRTC is not used anymore. Following that, the slicing scheme gradually increases the resources of the UE 3 until they reach a climax of 36% between approx. 130 to 140 secs in an effort to cope with the drop in the link quality (which is at the lowest level). Generally, the AI-unit assists in the advancement of the QoE since the WebRTC Throughput is increased from 0.5 to 4 Mbps and the Jitter drops from 10 to 5 ms during the second part of the experiment.

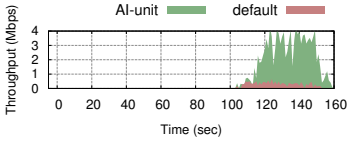
Overall, the QoE of all UEs is clearly enhanced given that the Throughput and Jitter performances are ameliorated. Moreover, the slices are provided in a sophisticated way so as to avoid over- and underprovisioning. In fact, this is illustrated in Fig. 5.9ε′, 5.10ε′, 5.11ε′. The red line depicts a value of 33%, which would be the highest slice that could be allocated by a UE with a fixed and fair slicing algorithm. Importantly, our dynamic scheme is able to surpass this limit when the demand for resources is extremely large as well as to decrease the resources dramatically lower than this percentage when the connection quality is excellent giving, this way, the chance for link improvement to other UEs in the network.

5.4.3 Online - Distributed Training

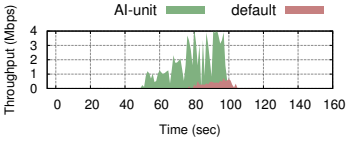
To evaluate the MLOps architecture, we run scenarios with new traffic patterns. In specific, we slightly altered the noise distributions in the augmentation steps (sect. 5.3.2) for the new scenarios. In the steps where the standard normal distribution was utilized, we replaced it with an AWGN with a mean of 0 and sd of 1.5. Moreover, we replaced the AWGN with a mean of 0 and an sd of 10 with a new distribution of the same mean but an sd of 15. Thus, we represent a small change in the distribution of the traffic pattern since the baseline patterns still exist in the new scenarios. We noticed that as soon as the new traffic patterns arrived, the predictions deviated quite a bit and the accuracy dropped immediately below the predefined threshold (70%) as shown in Fig. 5.12. Then, the Kubeflow Pipeline was trig-



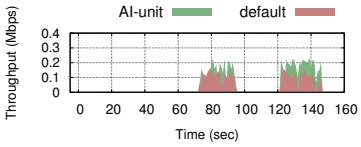
(α′) UE 1 interaction with WebRTC service



(α′) UE 2 interaction with WebRTC service

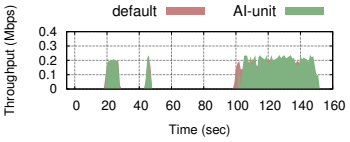


(α′) UE 3 interaction with WebRTC service

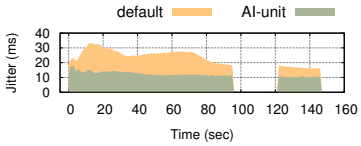


(β′) UE 1 interaction with Nginx service

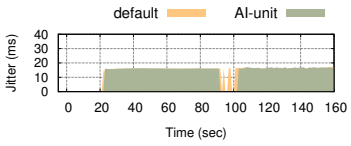
(β′) UE 2 interaction with SIPp service



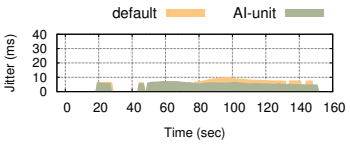
(β′) UE 3 interaction with Nginx service



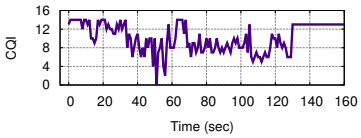
(γ′) UE 1 experienced Jitter



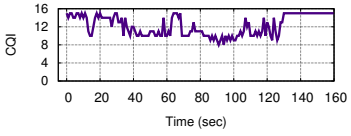
(γ′) UE 2 experienced Jitter



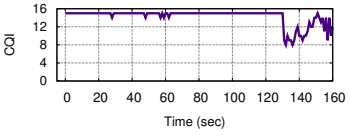
(γ′) UE 3 experienced Jitter



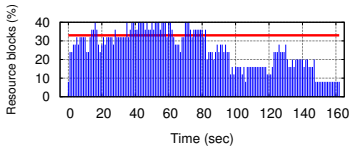
(δ′) UE 1 experienced CQI



(δ′) UE 2 experienced CQI

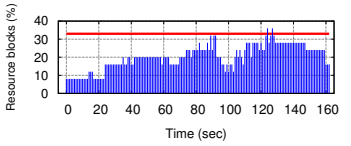


(δ′) UE 3 experienced CQI



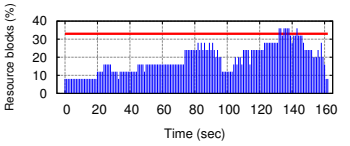
(ε′) UE 1 allocated slices. The red line indicates a fair sharing scheme with 33% of resources.

Figure 5.9: UE 1 QoE with and without the AI unit equipped with CNN-LSTM.



(ε′) UE 2 allocated slices. Red line indicates a fair sharing scheme with 33% of resources.

Figure 5.10: UE 2 QoE with and without the AI unit equipped with CNN-LSTM.



(ε′) UE 3 allocated slices. The red line indicates a fair sharing scheme with 33% of resources.

Figure 5.11: UE 3 QoE with and without the AI unit equipped with CNN-LSTM.

gered and started the process of distributed training. In between, the slicing decisions were defined reactively. After the training was over, the updated model started to make predictions again with high accuracy. Noticeably, it converged quite fast with approximately only 20 new samples-scenarios (50 minutes of receiving new samples and updating the model in real-time). It is fast since 300 samples were used for the offline training. Overall, the ability

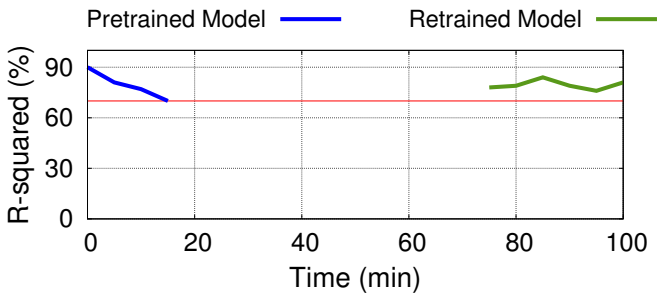
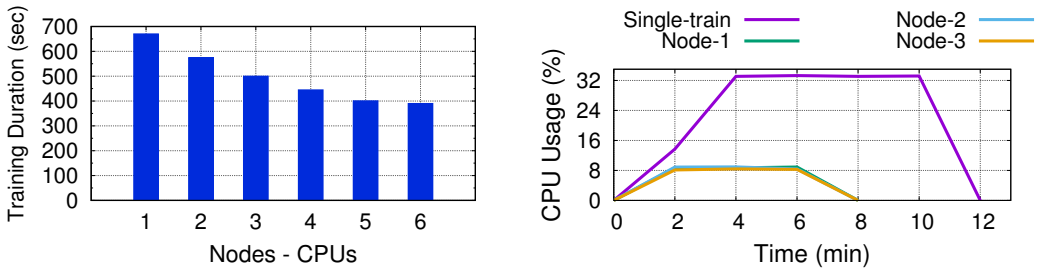


Figure 5.12: Error before & after online training. The red horizontal line indicates the error threshold.

of the scheme to cope with the new patterns relies on many components. First, the differences between the new pattern distribution with the one that the model has converged previously. The bigger the difference the larger the number of new samples required. Further, the processing power of the infrastructure is vital. For instance, Graphics Processing Units (GPUs) and TPUs outperform CPUs substantially accelerating the updating.

To evaluate our distributed training architecture we scale our cluster up to six NITOS nodes that carry octa-core processors (Intel-Core i7-3770 at 3.40 GHz Processor). Observing Fig. 5.13α', the increase in performance is almost linear as the training time seems to converge at 6 CPUs succeeding in reducing training time by half. This optimization of training time enables us to train the model as quickly as possible and to be able to cope more accurately with the predictions of the most recent data of traffic and network conditions. It is worth noting that the training data were taken from a sample of the entire dataset: 20 scenarios with 18 columns-features. The distributed training is applied to our cluster (NITOS Testbed) where only CPUs are used and the purpose of this experiment was to show how beneficial it is to use all resources simultaneously in the case of online training. The CNN-LSTM model was employed for the experiment. Performance can be further enhanced by utilizing a GPU cluster. In addition, load balancing is ensured in our cluster as illustrated in 5.13β'. In this experiment, we compared the CPU usage for the training of the model between a single machine-container and distributed 3 pods - 3 nodes synchronous all-reduce training. We notice that the single pod has almost 4 times CPU usage compared to the distributed pods which consume resources evenly in the cluster. These measurements were taken from the Prometheus adapter, which we integrated into the cluster for resource monitoring.



(α′) Dist. Training Duration (β′) Dist. CPU Usage

Figure 5.13: Experimental results for Distributed Training

5.5 Limitations and Discussions

While our results add valuable insights to the evolving domain of slicing in cloud-native 5G Networks, it's important to recognize the limitations of our infrastructure. The constraint on the number of UEs, capped at three, was a practical consideration due to the challenges associated with establishing connections in our real telecommunications network setup. The setup operates as a private 5G network where the application usage is more static, meaning the variety of applications that the users interact with, is relatively fixed. This may not fully represent the dynamic nature of application usage in public 5G networks, where applications with different network requirements may be in use simultaneously. To address this, extensive datasets that capture a wide range of user behaviors, application interactions, and network patterns are essential. These datasets will serve as the foundation for training machine learning models and refining the slice allocation algorithm to handle the intricacies of dynamic application usage in public 5G networks. Also, by increasing the scale of the experimental setup by connecting a larger number of end devices is crucial to emulate the complexities of public networks. This expansion allows for a more comprehensive evaluation of the slice allocation mechanism's performance in diverse and dynamic scenarios. Nevertheless, the service-aware slice allocation mechanism provides an end-to-end solution that can be directly plugged into any type of telecommunication network, regardless of the operator. From a performance perspective, there can be limitations concerning the real-time packet inspection and classification, as the overall cell throughput is increased. Such limitations can be easily overcome, when employing data-plane traffic accelerators in the network, for bypassing the operating system stack and providing direct access to the network. Implementations of libraries such as DPDK, enhanced Berkeley Packet Filters (eBPF) or employing a Vector Packet Processing (VPP) methodology in the packet handling can offer significant gains in performance, espe-

cially in the cases where the overall network traffic reaching the UPF surpasses 1Gbps. The aforementioned limitations provide avenues for future work, including extending the experimentation to larger-scale setups, exploring the performance of the slice allocation mechanism in public 5G networks with dynamic application usage, and investigating solutions to handle multiple UEs.

5.6 Conclusion

In this chapter, we developed and experimentally evaluated an ML-driven approach for defining the optimal slice application in the cellular 5G network, based on the applications that are hosted on top. Our framework can autonomously decide on the allocations, based on the ML-driven classification of the traffic and the mobility of users, providing near-real-time performance. The selection of the ML model was determined after experimenting with several neural network-based approaches, with the one performing optimally being a CNN-LSTM stacked model for our data. The solution is able to analyze and classify traffic from different applications correctly. At the same time, it considers the user's connection quality, and appropriately enforces the slices in the network. In the future, we foresee wrapping parts of our contribution into xApps and porting our solution to the O-RAN architecture. The detailed implementation instructions and code repository can be accessed on GitHub: GitHub.[1]. Additionally, partial datasets and code configurations for the framework are provided in [124].

[1]For specifics on the experimental setup, refer to: `https://github.com/teo-tsou/app_aware_5g`

Chapter 6

AI-Driven Attack Mitigation using Slicing for 6G Networks

6.1 Introduction

The advent of the 5th generation of mobile networks marks a transformative era in telecommunications unlocking countless opportunities for developing cutting-edge applications. However, a notable challenge persists in the absence of dynamic mechanisms for resource sharing among end users. This deficiency blocks the achievement of Key Performance Indicators (KPIs), often falling short due to under-provisioning. Furthermore, the deficiency in network optimization results in energy wastage, characterized by over-provisioning.

Additionally, network security is emerging as a critical concern, particularly with the increasing sophistication of network intrusion methods. Malicious attacks violate data integrity and privacy and significantly impact network performance. A network attack such as Denial of Service (DoS) can cause 5G core network functions such as the User Plane Function (UPF) to fail and even cause the Radio Access Network (RAN) to malfunction [127]. The effects of these security threats extend beyond networking failures. They introduce inefficiencies in the use of resources, increase operating costs, and require recovery efforts. Furthermore, monolithic-closed telecommunications infrastructures often lack the adaptive mechanisms to dynamically manage these threats, leading to vulnerabilities.

An effective strategy for optimizing resources and improving network performance involves classifying users and the traffic they generate. By identifying and classifying traffic patterns, more efficient resource allocation is possible, ensuring that legitimate users receive

the necessary bandwidth while mitigating the impact of malicious activities. The OpenRAN (O-RAN) architecture [128] offers a fertile ground for such strategies, as it exposes the RAN functionalities by controlling them through Radio Intelligent Controllers (RIC) via open interfaces. O-RAN's architecture enables a strategic logic guiding network optimizations through a three-step process: infer, decide, and determine. By integrating AI/ML, we can infer the network's current state and take the appropriate actions in real-time. This allows us to analyze past network behaviors, make informed decisions, and determine the most effective actions to optimize resource allocation and enhance security.

In this chapter, we adopt this three-step process by proposing an end-to-end 5G innovative framework that leverages AI/ML techniques to classify network traffic in real-time and dynamically adjust resource allocation and user management within the O-RAN architecture. Specifically, we developed an intrusion detection xApp that utilizes AI/ML models, trained on real-world datasets, to classify user traffic and make appropriate slicing and user management decisions on Radio Resource Control (RRC) level within the network. Our framework is a real-world solution, developed and tested on OAI [38] using standardized O-RAN interfaces and Service Models (SM). The ultimate goal of our solution is to suppress the network attacks and to maintain and even enhance the user experience during such incidents.

6.2 Related Work

The development of Network Intrusion Detection Systems (NIDS) has been studied extensively, with a significant focus on integrating AI/ML techniques to improve detection accuracy. A comprehensive survey in [129] underlines the importance of integrating machine learning algorithms into network anomaly detection systems, providing an in-depth review of Supervised Learning (SL) and Reinforcement Learning (RL) models. In [130], the authors proposed a Deep Learning-based (DL) self-adaptive architecture for anomaly detection, demonstrating the system's capability to handle fluctuating network traffic and achieve efficient anomaly detection performance. Similarly, [131] achieved high accuracy scores by converting network flows into images for analysis by a Convolutional Neural Network (CNN). Furthermore, Federated Learning (FL) architectures have been introduced to NIDS for cloud-native 5G Networks [132], showcasing the benefits of distributed learning in maintaining data privacy. Many works also focused explicitly on DoS attacks by propos-

ing DL strategies and architectures for O-RAN 5G networks [133] [134] [135]. These studies highlight the importance of AI/ML in identifying and suppressing such attacks, although they often rely on simulated environments that may not reflect real-world complexities. Towards integrating these models into actual 5G and O-RAN networks, [136] designed an early detection system for DoS attacks using a custom RIC in srsRAN [137], yet it lacked mechanisms for subsequent network actions post-classification. A more holistic approach is presented in [138], where attacks are classified with high accuracy over the air using OSC-RIC [139] in an LTE testbed, and countermeasures are deployed to maintain low latency.

Regarding inference and resource allocation in O-RAN networks, [140] proposed an RL-based slicing framework to reduce Service Level Agreements (SLA) violations, evaluated within the OpenRAN Gym [141]. Similarly, a DL-based service-aware slicing scheme in [142] demonstrated high user experience and QoS within the OAI platform. The FlexSlice framework introduced in [105] involves redesigning the MAC scheduler for multi-level resource allocation, showing significant improvements in dynamic RAN slicing. Moreover, [143] presented an end-to-end O-RAN control loop for radio resource allocation in SDR-based 5G networks, focusing on real-time adaptability and resource efficiency through AI-driven xApps. In [144], authors leveraged RL for enabling 5G Dynamic Time Division Duplexing (TDD) within the O-RAN framework, achieving reduced latency.

Although these studies present advanced solutions for NIDS and RAN control/slicing, they do not integrate both anomaly detection and dynamic resource allocation in real-world environments. Existing works either focus on anomaly detection without subsequent network actions to mitigate detected anomalies or implement RAN control/slicing without considering real-time anomaly detection. Moreover, most NIDS solutions are based on simulations or assume the availability of full features during testing, which may not reflect the constraints of real-world systems.

In this chapter, we address this gap by designing, implementing, and evaluating a real-time network intrusion detection xApp within the O-RAN framework. Our solution combines real-time anomaly detection, dynamic resource allocation, and user management in a real-world setup. Specifically, our xApp employs AI/ML models trained on real-world datasets to classify network traffic and dynamically adjust resource allocation and user management. It identifies malicious users and triggers RRC connection release to mitigate their impact on the network, while prioritizing legitimate users through end-to-end slicing. By integrating

our solution into the OAI platform and leveraging standardized O-RAN interfaces and Service Models (SM) from FlexRIC, we demonstrate a practical implementation that enhances network security and efficiency.

Our overall setup is illustrated in Fig. 6.1 and consists of an end-to-end O-RAN 5G Network based on OAI and FlexRIC. The target facility used for the development, application, and evaluation of the AI-driven NID O-RAN 5G network is the NITOS testbed, which is part of SLICES-RI [145]. The deployment specifications are summarized in Table 6.1. Below we outline the main components of the solution that enable the continuous classification of traffic and the subsequent slicing and user connectivity management that seamlessly enables high QoE to the end-users. A video demonstration of the experiment setup is provided in the following link[1], while the experiment can be reproduced, by following the instructions and deploying the code available in Github [2].

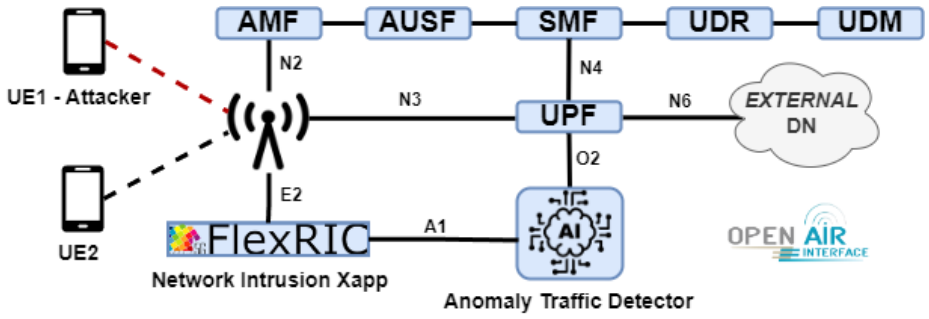


Figure 6.1: Experimental Setup: End-to-End Deployment of the AI-Driven Network Intrusion Detection 5G Network.

6.2.1 General Architecture and Management of the network functions

Starting from the Core Network functions, we relied on OAI's implementation and deployed them as microservices utilizing Docker containers. These functions include the basic 5G core components such as AMF, AUSF, SMF, UDR, UDM, and multiple UPFs with the different Single Network Slice Selection Assistance Information (S-NSSAI) values configured. An S-NSSAI configuration contains a Slice Service type (SST) and Slice Differentiator (SD). This enables a full end-to-end slicing as a UE may access multiple slices over the same

[1]Video demonstration available: `https://youtu.be/4hx1mAvhXMY`

[2]Link to reproducing the experimental setup: `https://github.com/teo-tsou/oai-anomaly-detection`

Table 6.1: Experimental Setup

System	Description
CPU	Intel(R) Core(TM) i7-14700 @ 2.101 GHz
Cores	20
GPU	NVIDIA GeForce RTX 4070
RAM	32GB
Operating System	Ubuntu 22.04.2 LTS
5G-Core Network	OAI v2.0.1
5G-RAN/E2-AGENT	OAI v2.0.0
O-RAN RIC	FlexRIC dev
O-RAN SM	RC v01.03
5G-UE	OAI v2.0.0
Packet Manipulator	Scapy
Dataset	KDDCUP'99 [146]
ML Library	TensorFlow

gNB. Each slice may serve a particular service type with an agreed SLA. Since the user traffic is passing through the GTP tunnels in the UPFs, the UPF is a critical point for classifying malicious user behaviors and identifying the user demands. 3GPP underlined the importance of the core traffic by standardizing the Network Data Analytics Function (NWDAF) function which mines the core data statistics and analyzes them. Considering that there is no integration of NWDAF on the O-RAN standardized architecture, we propose that NWDAF could be placed on the non-RT RIC and parse the user data through the O1 interface as the core network functions could be deployed on a Service Management and Orchestration framework. Subsequently, the NWDAF can enforce policies and send useful traffic summaries via the A1 interface to the RT RIC controller, which can then apply policies to the RAN through an xApp.

Since there is not yet an open implementation of the NWDAF functionality, we implemented our custom solution, the Anomaly Traffic Detector (ATD). This network function analyzes the traffic on the UPFs by leveraging a packet manipulator which in our case is

Scapy [147]. In our scheme, the ATD plays the role of the NWDAF. We also employ the RT RIC from FlexRIC. We utilize FlexRIC since it has the least overhead compared to most RIC implementations [40] and because it is O-RAN-compliant providing an E2 agent, nearRT-RIC, and an xApp SDK framework. Therefore, our E2-Agent is OAI's gNodeB, and the xApp is an application we developed utilizing FlexRIC's SDK to infer the RAN Functionalities of E2-Agent utilizing mainly the RC SM.

6.2.2 Dataset and Machine Learning

The ATD network function, beyond analyzing the user data on the UPF side, it incorporates an intelligent mechanism to classify malicious and normal traffic, by utilizing an ML model that is part of its architecture.

This ML model was trained on a real-world dataset KDDCUP'99 which is the most widely used data set for the evaluation of network intrusion systems [146]. The dataset contains a substantial number of instances, with over 4 million for training and around 311,029 for testing. The dataset includes a huge variety of features related to the basic network connection characteristics such as packet header information and advanced features such as content-related information. We selected five important features: the protocol type which defines the protocol used in the connection (TCP, SCTP, UDP), the service that is running in the destination network such as HTTP, FTP, or SSH, the flag which establishes the status of the connection such as normal (SF), rejected (REJ), or reset (RST), and finally the source and destination bytes. These features were chosen for their relevance in distinguishing between normal and malicious traffic and their compatibility with real-time analysis through Scapy, which extracts information only at the packet level. Furthermore, the dataset contains 4 attack labels: Probing Attack, Remote to Local Attack, Denial of Service Attack, and User to Root Attack. Our preprocessing steps are described below:

- Label Conversion: Converting multi-class labels into binary labels: 1 for any attack, 0 for normal.
- Flag Conversion: Converting the dataset's flag values to a format compatible with Scapy.
- Feature Selection: Selecting the Scapy-related/relevant features.

- Encoding and Scaling: To standardize the data we use OneHotEncoder for categorical features and MinMaxScaler for numerical features.

After preprocessing, we trained and excessively evaluated several AI/ML learning models using the TensorFlow implementation, including Random Forest, One-Class SVM, Local Outlier Factor, KNN, and Autoencoders. The comparison of the models led us to employ the Random Forest model due to its better performance compared to the other models in terms of accuracy and training/inference times.

6.2.3 Anomaly Detection and Countermeasures

The detailed operation of our framework is illustrated in Fig. 6.2. The ATD unit utilizing Scapy, continuously monitors UPF traffic and classifies clients based on their IP and S-NSSAI values. It manipulates each packet in real-time, extracting the necessary features that our ML model was trained on. After collecting the first N packets, the ATD preprocesses these features and feeds them into the Random Forest classifier. Then the Random Forest by applying a sliding window mechanism processes N=30 packets at a time, classifying the traffic as benign or malicious. The reason we selected 30 packets-window is to reduce infer/prediction times as close to real-time and avoid false outliers in the classification with a larger input range. Finally, the ATD sends the anomaly percentage per UE to the xApp for the RAN Control and countermeasures.

The functionality of the xApp is summarized in Algorithm 4. First, it connects to FlexRIC's RT RIC and subscribes to the RC SM offered by the E2 Agent. Then accepts incoming socket connections from ATD clients and initializes the necessary data structures for UE identification. In the main loop, it listens for ATD messages, and for each message, it extracts the UE ID, the S-NSSAI values, and the anomaly ratio per UE and updates the UE-related data structures. Then it determines the physical RB allocation based on the anomaly ratios it receives per UE, with the formula given by Eqn. 6.1. To avoid the total RB allocation exceeding 100%, it scales down proportionally the allocation through Eqn. 6.2 and 6.3. Finally, when a UE's anomaly ratio reaches 100%, the xApp classifies it as an attacker and triggers an RRC UE Connection release, causing the UE to disconnect from the network. Subsequently, the xApp reassigns the PRB allocation to the remaining UEs, ensuring efficient resource distribution.

$$\text{PRB}_i = (1 - \text{AnomalyRatio}_i) \cdot 100 \tag{6.1}$$

$$\text{ScalingFactor} = \frac{100}{\text{TotalPRB}} \tag{6.2}$$

$$\text{AdjustedPRB}_i = \text{PRB}_i \cdot \text{ScalingFactor} \tag{6.3}$$

Algorithm 4: xApp Functionality

Initialization:

Connect to RT-RIC and subscribe to the RC SM.

Set up socket and accept connections from ATD clients.

Initialize `ue_data` structure with default values.

while *True* **do**

Receive and Process Messages:

Listen for incoming messages from ATD clients.

Parse messages to extract UE ID, S-NSSAI, and anomaly ratios.

Update `ue_data` structure.

Traffic Analysis and Classification:

Determine PRB allocations based on anomaly ratios.

Resource Allocation and Enforcement:

if *a UE is an attacker (100% anomaly ratio)* **then**

Set PRB allocation to 0% for the attacker.

Distribute remaining PRB among other UEs.

Trigger RRC release for the attacker UE.

else

Adjust PRB allocations to ensure total does not exceed 100%.

Apply slicing changes to enforce PRB allocations.

end

end

6.3 Experimental Evaluation

To evaluate our solution, we first compared the performance of the ML models. Then, we tested the efficiency of our solution in both the preservation and enhancement of network

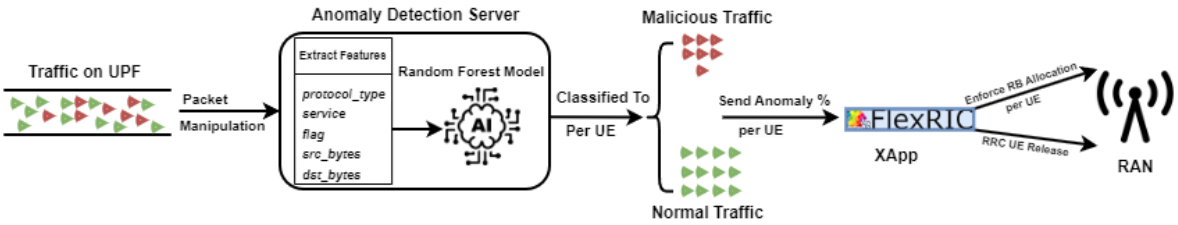


Figure 6.2: Detailed Architecture of the AI-Driven Network Intrusion Detection System.

performance and user experience during different anomaly scenarios, including DoS attacks.

We measured the ML model's performance using three metrics: accuracy, ROC AUC, and F1 scores. Accuracy measures the classified instances among all cases, calculated using Eqn. 6.4, where *TP* stands for True Positives, *TN* for True Negatives, *FP* for False Positives, and *FN* for False Negatives. ROC AUC distinguishes between classes by plotting the true positive rate (TPR) against the false positive rate (FPR). It is calculated by Eqn. 6.5. The F1 Score is the harmonic mean of precision and recall. It is calculated by Eqn. 6.6, where *Precision* is $\frac{TP}{TP+FP}$ and *Recall* is $\frac{TP}{TP+FN}$. We can observe from Fig. 6.3 that the Random Forest model had the best performance with high accuracy, ROC AUC, and F1 scores, making it the most reliable for our NIDS. The autoencoder also had similar performance but required more computational resources. Additionally, we measured the training and inference times for the different models, as shown in Table 6.2. The inference times correspond to the 30-packet window predictions. The Local Outlier Factor model achieved the shortest times for both training (0.77 sec) and inference (0.6 ms). However, due to its lack of accuracy, we explored the Random Forest model, which had a low training time (4.36 sec) and inference time (3.4 ms), making it suitable for our real-time system. On the other hand, the Autoencoder and KNN models had significantly longer training and inference times, which may limit their practical applicability in dynamic network environments such as ours. With these observations, we focused on the Random Forest Model due to its balanced performance and efficiency. Fig. 6.4 presents the confusion matrix for the Random Forest model, where 89.46% of benign traffic and 80.37% of attack traffic were correctly classified during testbed integration. These percentages, although lower compared to some literature, demonstrate the efficacy of our classifier under the constraints of real-world system integration. The limited feature set used, which is compatible with real-time analysis through Scapy, affects the overall accuracy. Despite these limitations, our framework achieves relatively high accuracy.

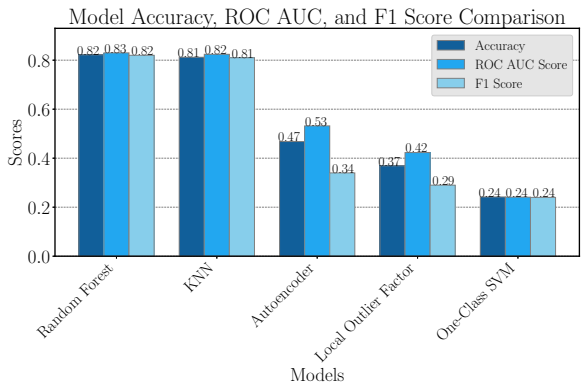


Figure 6.3: Model Training Evaluation: Accuracy, ROC AUC, and F1 Score Comparison.

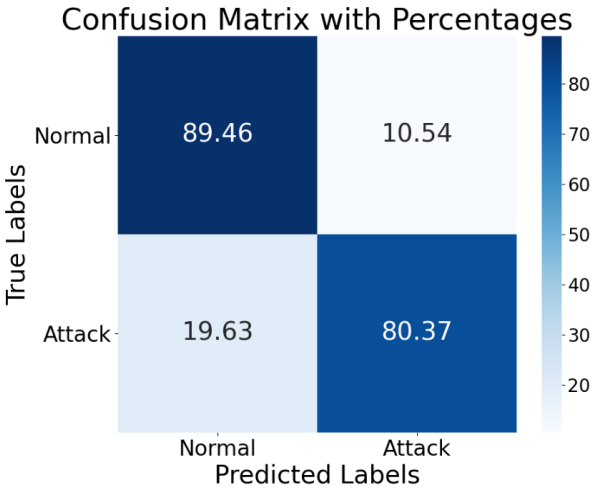


Figure 6.4: Confusion Matrix for the Random Forest Model.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.4}$$

$$\text{ROC AUC} = \int_0^1 \text{TPR}(\text{FPR}) \cdot d(\text{FPR}) \tag{6.5}$$

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.6}$$

To evaluate our solution under realistic conditions, we designed several traffic scenarios in our testbed connecting two UEs. In the initial scenario illustrated in Fig. 6.5, in the beginning, both UEs share roughly the same throughput since they slice equal amounts of RBs and both generate normal traffic. At the marked point with a red dotted line, UE1 begins to

Model	Train Time (s)	Infer Time (ms)
Random Forest	4.36	3.4
One-Class SVM	125.98	6.14
Local Outlier Factor	0.77	0.6
KNN	5.51	9.14
Autoencoder	181.46	23.11

Table 6.2: Training and inference times for various machine learning models

send some malicious packets into the network, generated via Scapy using the test (unseen) part of the KDDCUP'99 dataset. Due to the absence of any NID mechanisms in this scenario, both UEs continue to share the same network resources even after the introduction of malicious traffic, leading to a noticeable degradation in performance for the normal user, UE2. In the subsequent scenario in Fig. 6.6, we introduce our NID solution alongside the xApp. As soon as the malicious packets are inserted into the network, our system successfully classifies them as abnormal and relocates the RBs based on the anomaly percentage per UEs for every sliding window of 30 packets. Consequently, the QoE for the legitimate UE (UE2) improves significantly as it gets the most RBs, while the QoE for the suspicious UE (UE1) declines.

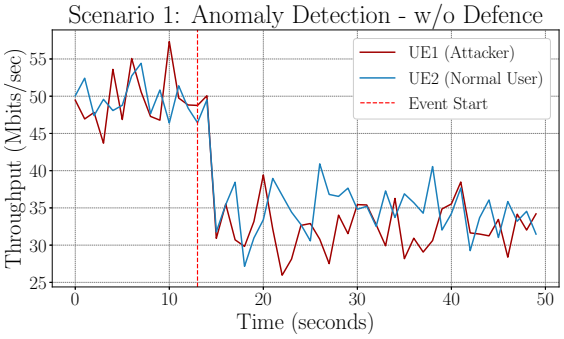


Figure 6.5: Anomaly Traffic w/o Defence.

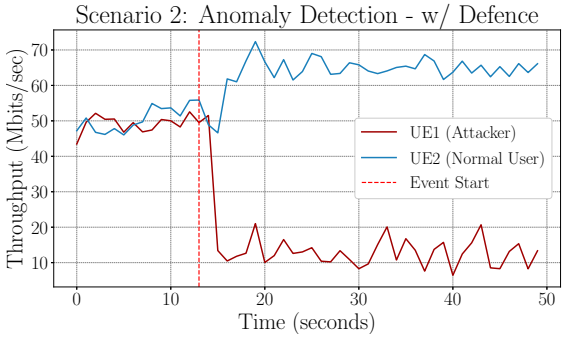


Figure 6.6: Anomaly Traffic w/ Defence.

Figure 6.7: Anomaly Traffic w/ and w/o Defence; red line denotes when the anomaly traffic was generated.

The third scenario demonstrated in Fig. 6.8 explores the network's vulnerability to a DoS attack, executed from UE1 using Hping3, without any defensive actions. During this attack, both UEs experience a dramatic drop in throughput to nearly zero. This illustrates the impact of the DoS attack across the 5G network without any defensive mechanism. In the final

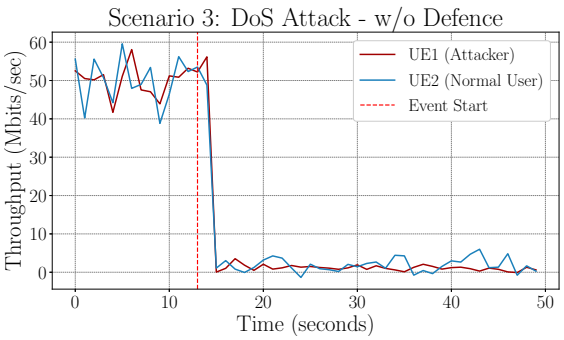


Figure 6.8: DoS Attack w/o Defence.

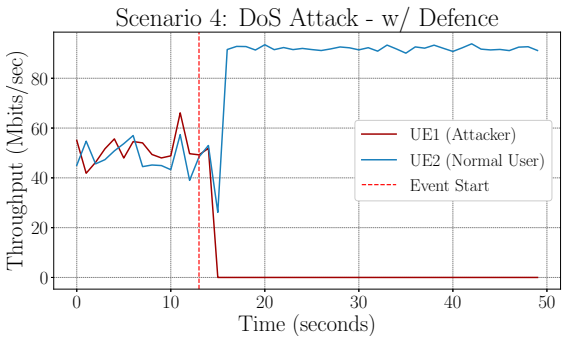


Figure 6.9: DoS Attack w/ Defence.

Figure 6.10: DoS Attack w/ and w/o Defence; red line denotes when the attack started.

scenario (see Fig. 6.9), we evaluate the resilience of our system to the same DoS attack, but this time with our defensive solution activated. Our system identified the DoS attack since the anomaly percentage of the 30-packet sliding window reached a 100% threshold. Then the xApp triggers an RRC connection release specifically for UE1. This isolates the attacking UE and prevents it from further degrading network performance. UE2 experiences minimal disruption, although its throughput almost doubled and remained largely stable, illustrating the system's effectiveness in detecting and actively preventing sustained network attacks. This ensures that normal network users maintain QoE even under attack conditions.

Furthermore, we measured the impact of the DoS attack on normal user latency. Specifically, we measured the Round Trip Time (RTT) for UE2 under attack conditions, both with and without our defense mechanisms, as illustrated in Fig. 6.11. The results indicate a significant rise in RTT during the attack, with latency peaking at nearly 3 sec without our NIDS. When our xApp is running, the RTT remains substantially lower, maintaining an average latency of approximately 18 ms. That indicates that our framework successfully suppressed the attack and kept the user's latency at a low level.

Finally, our solution demonstrates significant energy efficiency improvements. As depicted in Fig. 6.12, we monitored the CPU usage within the UPF docker container during a DoS attack, both with and without our NIDS solution. With our defense mechanism, we observed an average reduction of up to 15% in CPU usage, reversing the failure of UPF and enhancing energy savings.

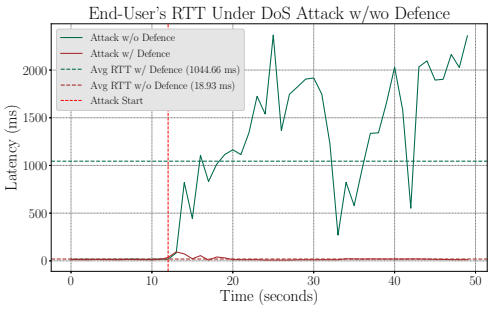


Figure 6.11: End-User's RTT Under DoS Attack w/wo Defence.

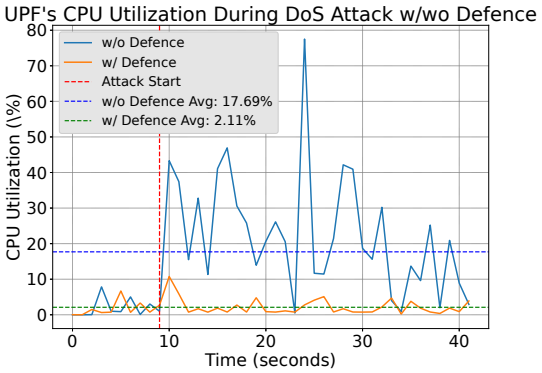


Figure 6.12: UPF's CPU Utilization During DoS Attack w/wo Defence.

6.4 Conclusion

In this chapter, we have demonstrated an AI/ML-driven framework for Network Intrusion Detection and dynamic resource/user management within the O-RAN architecture, focusing on enhancing network security and optimizing 5G network performance. Our solution effectively handles anomaly traffic and mitigates intrusion methods such as Denial of Service (DoS) attacks through real-time traffic classification and dynamic slicing and RRC UE connection management, by extending the RC SM in FlexRIC. Through extensive evaluation of various AI/ML models, the Random Forest model was selected for its balanced performance and efficiency. Despite the limitations of our system, which include relying on available features during packet manipulation that can affect the model's accuracy, our solution still achieved good accuracy rates. Our experimental results highlight the benefits of our framework: maintaining low latency under attack conditions, nearly doubling throughput for legitimate users, and reducing CPU usage by up to 15%. In the future, we foresee extending our solution by dynamically managing/relocating edge computing resources (e.g. Multiple Access Edge Computing services) based on the anomaly detection mechanisms of our system.

Chapter 7

Conclusions

7.1 Summary of Contributions

This thesis addressed critical research challenges related to the management and operation of emerging 6G networks, particularly focusing on dynamic resource allocation, mobility-aware service continuity, and AI-driven network security. Our contributions extend across four main pillars: *Cloud-Native MEC Migration*, *Deep Reinforcement Learning-driven MEC Orchestration*, *Service-Aware Network Slicing with Machine Learning*, and *AI-Powered Intrusion Detection and Mitigation within the O-RAN Architecture*.

In Chapter 3, we proposed a novel cloud-native *Follow-me MEC* framework, emphasizing the concept of placing MEC services within the fronthaul segment of a disaggregated heterogeneous base station architecture. This strategic placement significantly reduced end-to-end latency, a crucial improvement for latency-sensitive 6G services. Leveraging Kubernetes for management and KubeVirt for integrating Virtual Machines, we successfully demonstrated seamless and automatic live service migrations triggered by changing network conditions and UE mobility. The experimental evaluation validated our scheme's ability to dynamically switch between radio access technologies and infrastructure nodes without user-perceived interruptions, thus maintaining stringent service-level agreements even in highly dynamic scenarios.

Building upon this foundation, Chapter 4 introduced an advanced mobility-aware edge infrastructure that leverages artificial intelligence techniques—specifically Deep Reinforcement Learning —to proactively orchestrate MEC services. We implemented a digital-twin simulation environment mirroring our real-world setup, facilitating the training of DRL agents

(DQN and DSQN) without disrupting production environments. The trained agents utilized multi-cell RTT metrics and edge node workloads to proactively trigger migrations, significantly improving the system's responsiveness. Real-world experimental validations confirmed that our DQN-based agent efficiently learned complex migration strategies, consistently delivering low latency and uninterrupted high-throughput experiences, outperforming static migration schemes by considerable margins.

In Chapter 5, we tackled the increasingly relevant challenge of dynamic and service-aware network slicing. Recognizing the limitations of static slicing approaches, we proposed a sophisticated ML-driven solution employing time-series deep learning models. After rigorous comparative evaluations of several architectures—including CNNs, LSTM networks, GRUs, and hybrid CNN-LSTM stacks—we identified the CNN-LSTM stacked architecture as superior for our datasets. By accurately predicting user-application interactions and network conditions (e.g., mobility-driven fluctuations in signal quality), our framework dynamically adjusted slice configurations in near real-time, significantly improving resource efficiency and user experience consistency. Additionaly, we provided extensive experimentation on the model's lifecycle by designing and developing a distributeed MLOps architecture. Detailed performance analyses and open-source release of our experimental codebase further strengthen the reproducibility and practical applicability of our findings.

In Chapter 6, we addressed the critical challenge of security within softwarized and AI-driven network architectures. Recognizing the vulnerability of 6G networks to increasingly sophisticated threats such as DoS attacks, we integrated an AI-driven intrusion detection framework within the O-RAN architecture, specifically utilizing FlexRIC to implement an adaptive xApp-based intrusion detection and mitigation scheme. We conducted extensive comparative evaluations across various ML classifiers, ultimately selecting the Random Forest model for its superior balance of accuracy, computational efficiency, and real-time responsiveness. The proposed framework dynamically identified and mitigated network threats, automatically reallocating network resources and adjusting UE connections. Empirical results demonstrated the solution's capability to maintain high QoS under attack scenarios, substantially increasing legitimate user throughput, reducing latency degradation, and decreasing overall CPU utilization by up to 15%.

Collectively, these contributions represent a significant advancement toward the realization of autonomously managed, highly intelligent, and secure 6G networks. Our research di-

rectly supports the vision of future networks characterized by native AI integration, extreme service continuity under mobility, dynamic resource allocation responding to real-time conditions, and robust self-defending mechanisms against security threats.

7.2 Perspectives for Future Work

Building upon the contributions and insights gained through this thesis, there are many opportunities to explore for future works, addressing open questions and extending the capabilities of the proposed approaches.

In Chapter 3, we presented a cloud-native framework for seamless service migration in MEC both 3GPP and non-3GPP environments leveraging Kubernetes and virtualization technologies. A natural extension of this work involves exploring unified resource management and orchestration methods that jointly optimize MEC resource allocation and network slicing decisions. Integrating a predictive handover decision mechanism synergistically with MEC migrations, and evaluating these in more heterogeneous and large-scale environments (such as multi-domain deployments or edge-cloud federations), will further enhance service continuity and latency management for highly mobile users. Additionally, future research could explore energy-efficient migration techniques, balancing service performance and sustainability,objectives that are in line with 6G goals.

In Chapter 4, we introduced Deep Reinforcement Learning algorithms for proactive, mobility-aware MEC service migration. An interesting avenue for further investigation would be employing federated reinforcement learning approaches to address scalability, privacy concerns, and model generalizability across distributed edge nodes. Moreover, combining DRL migration decisions with semantic-aware networking strategies—where data relevance drives migration and network resource prioritization—could enhance user experience in future applications. A large-scale deployment in a real-world industrial scenario, for instance within autonomous vehicles or smart city infrastructures, would also validate the robustness and adaptability of the proposed approach.

In Chapter 5, we proposed an ML-based, service-aware network slicing MLOps framework utilizing CNN-LSTM architectures. Future work may extend this framework by exploring more advanced and explainable AI architectures, including transformer-based or attention-driven models, to improve prediction accuracy and interpretability of network slicing deci-

sions. Moreover, incorporating online learning approaches to handle real-time adaptation to changing user behavior, service demands, and radio conditions could significantly enhance the flexibility and efficiency of slicing mechanisms. Finally, standardizing this solution as a deployable xApp/rApp within O-RAN would support broad adoption and encourage industry-wide experimentation and validation.

In Chapter 6, we developed an AI-driven network intrusion detection and mitigation framework within the O-RAN context. Future extensions include exploring generative AI techniques to proactively identify previously unseen cyber threats and rapidly adapt mitigation strategies accordingly. Additionally, extending the anomaly detection capabilities through decentralized and federated learning would enhance the scalability, responsiveness, and security of large-scale deployments. Another promising research direction involves coupling the detection mechanism with dynamic resource provisioning and edge computing relocation strategies. Such integration could offer holistic security and resource optimization, significantly improving the resilience and operational efficiency of 6G network infrastructures.

7.2.1 Lessons Learned and Outlook

Reflecting on the entirety of this thesis, several key lessons emerge. Firstly, the integration of cloud-native principles with AI-driven network management has demonstrated powerful synergies, enabling highly adaptive, scalable, and resilient 6G infrastructures. Secondly, the importance of end-to-end experimental validation combining both digital twin environments and real-world trials has proven essential to assess feasibility and performance in practical deployments. Techniques that work convincingly in theory may face unexpected challenges when exposed to the complexities, uncertainties, and dynamic conditions of real-world systems. Thirdly, pursuing explainable and sustainable AI frameworks remains vital to transparency, and environmental responsibility in future networks.

Looking ahead, a major challenge is achieving a truly holistic orchestration framework capable of seamlessly coordinating mobility, security, slicing, and sustainability objectives under a unified management plane. The convergence of diverse 6G enablers — from semantic communications to zero-touch automation and beyond — will demand not only advanced technical solutions but also cross-disciplinary collaboration and standardization efforts. Addressing these challenges will be pivotal to realizing the vision of fully autonomous, trust-

worthy, and human-centric 6G networks.

References

[1] Theodoros Tsourdinis, Ilias Chatzistefanidis, Nikos Makris, Thanasis Korakis, Navid Nikaein, and Serge Fdida. Service-aware real-time slicing for virtualized beyond 5g networks. *Computer Networks*, 247:110445, 2024.

[2] Sonia Shahzadi, Muddesar Iqbal, Tasos Dagiuklas, and Zia Ul Qayyum. Multi-access edge computing: open issues, challenges and future perspectives. *J. Cloud Comput.*, 6(1), December 2017.

[3] Theodoros Tsourdinis, Ilias Chatzistefanidis, Nikos Makris, and Thanasis Korakis. Ai-driven service-aware real-time slicing for beyond 5g networks. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2022.

[4] Theodoros Tsourdinis, Nikos Makris, Serge Fdida, and Thanasis Korakis. Drl-based service migration for mec cloud-native 5g and beyond networks. In *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, pages 62–70, 2023.

[5] Theodoros Tsourdinis, Nikos Makris, Thanasis Korakis, and Serge Fdida. Demystifying urllc in real-world 5g networks: An end-to-end experimental evaluation. In *GLOBECOM 2024 - 2024 IEEE Global Communications Conference*, pages 2954–2959, 2024.

[6] Theodoros Tsourdinis, Nikos Makris, Thanasis Korakis, and Serge Fdida. Ai-driven network intrusion detection and resource allocation in real-world o-ran 5g networks. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '24, page 1842–1849, New York, NY, USA, 2024. Association for Computing Machinery.

[7] Nikos Makris, Virgilios Passas, Apostolos Apostolaras, Theodoros Tsourdinis, Ilias Chatzistefanidis, and Thanasis Korakis. On enabling remote hands-on computer networking education: the nitos testbed approach. In *2023 IEEE Integrated STEM Education Conference (ISEC)*, pages 132–138, 2023.

[8] Sokratis Christakis, Theodoros Tsourdinis, Nikos Makris, Thanasis Korakis, and Serge Fdida. Evaluation of user plane function implementations in real-world 5g networks. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2024.

[9] International Telecommunication Union (ITU). Minimum requirements related to technical performance for IMT-2020 radio interface(s). Technical Report ITU-R M.2410-0, ITU Radiocommunication Sector (ITU-R), November 2017. Accessed: 2024-03-31.

[10] Khaled B. Letaief, Wei Chen, Yuanming Shi, Jun Zhang, and Ying-Jun Angela Zhang. The roadmap to 6g: Ai empowered wireless networks. *IEEE Communications Magazine*, 57(8):84–90, 2019.

[11] Ericsson. Co-creating a cyber-physical world. Technical Report GFTL-24:000856 Uen, Ericsson, July 2024. Accessed: 2024-03-31.

[12] Li-Hsiang Shen, Kai-Ten Feng, and Lajos Hanzo. Five facets of 6g: Research challenges and opportunities. *ACM Computing Surveys*, 55(11):1–39, February 2023.

[13] 6G-IA Vision and Societal Challenges Working Group. Sustainability of 6g: Ways to reduce energy consumption. Technical report, 6G Smart Networks and Services Industry Association (6G-IA), November 2023. Accessed: 2024-03-31.

[14] Mikko Uusitalo, Patrik Rugeland, Mauro Boldi, Emilio Strinati, Gino Carrozzo, Panagiotis Demestichas, Mårten Ericson, Gerhard Fettweis, Marie-Helene Hamon, Matti Latva-aho, Josep Martrat, Aarno Parssinen, Bjorn Richerzhagen, Dario Sabella, Hans Schotten, Pablo Serrano, Giovanni Stea, Tommy Svensson, S. Soykan, and Yaning Zou. Hexa-x the european 6g flagship project. 06 2021.

[15] Ltd. Huawei Technologies Co. 6g: The next horizon, n.d. Accessed: 2025-09-29.

[16] 5G Infrastructure Association. European Vision for the 6G Network Ecosystem, June 2021.

[17] Wen Wu, Conghao Zhou, Mushu Li, Huaqing Wu, Haibo Zhou, Ning Zhang, Xuemin Sherman Shen, and Weihua Zhuang. Ai-native network slicing for 6g networks. *IEEE Wireless Communications*, 29(1):96–103, 2022.

[18] ETSI Industry Specification Group (ISG) Experiential Networked Intelligence (ENI). Experiential Networked Intelligence (ENI); Transformer Architecture for Policy Translation. `https://www.etsi.org/deliver/etsi_gs/ENI/001_099/030/04.01.01_60/gs_ENI030v040101p.pdf`, March 2024. ETSI GS ENI 030 V4.1.1.

[19] 3GPP. Study on new radio access technology: Radio access architecture and interfaces. Technical Report TR 38.801 V14.0.0, 3GPP, March 2017.

[20] Nikos Makris, Pavlos Basaras, Thanasis Korakis, Navid Nikaein, and Leandros Tassiulas. Experimental evaluation of functional splits for 5g cloud-rans. *IEEE Conference Record - International Conference on Communications*, 05 2017.

[21] Small Cell Forum. 5G nFAPI Specifications, November 2020. Document number: SCF225, Release 6.

[22] Daniel Dik and Michael Berger. Open-ran fronthaul transport security architecture and implementation. *IEEE Access*, PP:1–1, 01 2023.

[23] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. Technical report, ONF, 2012. Available online: `https://www.opennetworking.org/sdn-resources/sdn-library/whitepapers/`.

[24] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM*, 38(2):69–74, 2008.

[25] O-RAN Alliance. O-RAN Architecture Overview. Technical report, O-RAN Alliance, 2021. Available online: `https://www.o-ran.org/specifications`.

[26] 3rd Generation Partnership Project (3GPP). NG-RAN; Stage 2 functional specification of User Equipment (UE) positioning in NG-RAN. Technical Report TS 38.305, 3GPP, 2021. Available online: https://www.3gpp.org/ftp/Specs/archive/38_series/38.305/.

[27] Ericsson. 5G positioning: What you need to know. Technical report, Ericsson, 2020. Available online: https://www.ericsson.com/en/blog/2020/12/5g-positioning--what-you-need-to-know.

[28] Andrea Pinto, Giuseppe Santaromita, Claudio Fiandrino, Domenico Giustiniano, and Flavio Esposito. Characterizing location management function performance in 5g core networks. In *2022 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 66–71, 2022.

[29] ETSI MEC. Multi-Access Edge Computing: Architecture and Principles. Technical report, ETSI, 2021. Available online: https://www.etsi.org/technologies/multi-access-edge-computing.

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

[31] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[32] Etsi gs mec. https://www.etsi.org/deliver/etsi_gs/MEC/001_099/026/02.01.01_60/gs_MEC026v020101p.pdf.

[33] Nikos Makris, Virgilios Passas, Thanasis Korakis, and Leandros Tassiulas. Employing mec in the cloud-ran: An experimental analysis. pages 15–19, 10 2018.

[34] Shangguang Wang, Jinliang Xu, Ning Zhang, and Yujiong Liu. A survey on service migration in mobile edge computing. *IEEE Access*, 6:23511–23528, 2018.

[35] Serge Fdida, Nikos Makris, Thanasis Korakis, Raffaele Bruno, Andrea Passarella, Panayiotis Andreou, Bartosz Belter, CÃ©dric Crettaz, Walid Dabbous, Yuri Dem-

chenko, and Raymond Knopp. Slices, a scientific instrument for the networking community. *Computer Communications*, 193:189–203, 2022.

[36] Serge Fdida, Timur Friedman, and Sophia MacKeith. Onelab: Developing future internet testbeds. In Elisabetta Di Nitto and Ramin Yahyapour, editors, *Towards a Service-Based Internet*, pages 199–200, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[37] Nikos Makris, Christos Zarafetas, Spyros Kechagias, Thanasis Korakis, Ivan Seskar, and Leandros Tassiulas. Enabling open access to LTE network components; the NITOS testbed paradigm. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pages 1–6. IEEE, 2015.

[38] Florian Kaltenberger, Guy de Souza, Raymond Knopp, and Hongzhi Wang. The OpenAirInterface 5G New Radio Implementation: Current Status and Roadmap. In *WSA 2019; 23rd International ITG Workshop on Smart Antennas*, pages 1–5, 2019.

[39] Robert Schmidt, Chia-Yu Chang, and Navid Nikaein. FlexVRAN: A flexible controller for virtualized RAN over heterogeneous deployments. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[40] Robert Schmidt, Mikel Irazabal, and Navid Nikaein. FlexRIC: an SDK for next-generation SD-RANs. In *Proceedings of the 2021 17th CoNEXT*. ACM, 2021.

[41] Docker. `https://docs.docker.com/get-started/overview/`.

[42] Kubernetes. `https://kubernetes.io/docs/concepts/overview/components/`.

[43] KubeVirt: Virtualization extension for Kubernetes. [Online], `https://github.com/kubevirt/kubevirt`.

[44] Dejan Golubovic and Ricardo Rocha. Training and Serving ML workloads with Kubeflow at CERN. In *EPJ Web of Conferences*, volume 251, page 02067. EDP Sciences, 2021.

[45] Rubayet Shafin, Lingjia Liu, Vikram Chandrasekhar, Hao Chen, Jeffrey Reed, and Jianzhong Charlie Zhang. Artificial intelligence-enabled cellular networks: A critical path to beyond-5G and 6G. *IEEE Wireless Communications*, 27(2):212–217, 2020.

[46] Navid Nikaein, Eryk Schiller, Romain Favraud, Raymond Knopp, Islam Alyafawi, and Torsten Braun. *Towards a Cloud-Native Radio Access Network*, pages 171–202. Springer International Publishing, Cham, 2017.

[47] Osama Arouk and Navid Nikaein. 5G Cloud-Native: Network Management & Automation. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–2, 2020.

[48] Sameerkumar Sharma, Raymond Miller, and Andrea Francini. A cloud-native approach to 5g network slicing. *IEEE Communications Magazine*, 55(8):120–127, 2017.

[49] Kekki S. et al. ETSI White Paper No. 28: MEC in 5G networks, 2018.

[50] A. Reznik et al. ETSI White Paper No. 23: Cloud RAN and MEC: A Perfect Pairing, 2018.

[51] Giust F. et al. ETSI White Paper No. 24: MEC Deployments in 4G and Evolution Towards 5G, 2018.

[52] Nikos Makris, Virgilios Passas, Thanasis Korakis, and Leandros Tassiulas. Employing mec in the cloud-ran: An experimental analysis. pages 15–19, 10 2018.

[53] Nikos Makris, Virgilios Passas, Christos Nanis, and Thanasis Korakis. On minimizing service access latency: Employing mec on the fronthaul of heterogeneous 5g architectures. In *2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–6, 2019.

[54] Abdelkader Aissioui, Adlen Ksentini, Abdelhak Mourad Gueroui, and Tarik Taleb. On enabling 5G automotive systems using follow me edge-cloud concept. *IEEE Transactions on Vehicular Technology*, 67(6):5302–5316, 2018.

[55] Tung V Doan, Zhongyi Fan, Giang T Nguyen, Hani Salah, Dongho You, and Frank HP Fitzek. Follow me, if you can: A framework for seamless migration in mobile edge cloud. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1178–1183. IEEE, 2020.

[56] Tao Ouyang, Zhi Zhou, and Xu Chen. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. *IEEE Journal on Selected Areas in Communications*, 36(10):2333–2345, 2018.

[57] Nikos Makris, Christos Zarafetas, Pavlos Basaras, Thanasis Korakis, Navid Nikaein, and Leandros Tassiulas. Cloud-based convergence of heterogeneous rans in 5g disaggregated architectures. 05 2018.

[58] Cong Shen, Cem Tekin, and Mihaela van der Schaar. A non-stochastic learning approach to energy efficient mobility management. *IEEE Journal on Selected Areas in Communications*, 34(12):3854–3868, 2016.

[59] SIPp - a SIP protocol test tool. [Online], `https://github.com/SIPp/sipp`.

[60] Yaqiong Liu, Mugen Peng, Guochu Shou, Yudong Chen, and Siyu Chen. Toward Edge Intelligence: Multiaccess Edge Computing for 5G and Internet of Things. *IEEE Internet of Things Journal*, 7(8):6722–6747, 2020.

[61] Wei Lu, Xianyu Meng, and Guanfei Guo. Fast Service Migration Method Based on Virtual Machine Technology for MEC. *IEEE Internet of Things Journal*, 6(3):4344–4354, 2019.

[62] Zhihan Lv and Wenqun Xiu. Interaction of Edge-Cloud Computing Based on SDN and NFV for Next Generation IoT. *IEEE Internet of Things Journal*, 7(7):5706–5712, 2020.

[63] Marius Corici, Pousali Chakraborty, and Thomas Magedanz. A Study of 5G Edge-Central Core Network Split Options. *Network*, 1(3):354–368, 2021.

[64] Seungyeol Lee, Soohwan Lee, and Myung-Ki Shin. Low Cost MEC Server Placement and Association in 5G Networks. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 879–882, 2019.

[65] Mustafa Emara, Miltiades C. Filippou, and Dario Sabella. MEC-Assisted End-to-End Latency Evaluations for C-V2X Communications. In *2018 European Conference on Networks and Communications (EuCNC)*, 2018.

[66] I. Farris, T. Taleb, H. Flinck, and A. Iera. Providing ultra-short latency to user-centric 5G applications at the mobile network edge. *Transactions on Emerging Telecommunications Technologies*, 29(4):e3169, 2018. e3169 ett.3169.

[67] Shunmugapriya Ramanathan, Koteswararao Kondepu, Miguel Razo, Marco Tacca, Luca Valcarenghi, and Andrea Fumagalli. Live Migration of Virtual Machine and Container Based Mobile Core Network Components: A Comprehensive Study. *IEEE Access*, 9:105082–105100, 2021.

[68] Shunmugapriya Ramanathan, Abhishek Bhattacharyya, Koteswararao Kondepu, Miguel Razo, Marco Tacca, Luca Valcarenghi, and Andrea Fumagalli. Demonstration of Containerized Central Unit Live Migration in 5G Radio Access Network. In *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, pages 225–227, 2022.

[69] Hadeel Abdah, João Paulo Barraca, and Rui L. Aguiar. Handover prediction integrated with service migration in 5g systems. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–7, 2020.

[70] Marco Pomalo, Van Thanh Le, Nabil El Ioini, Claus Pahl, and Hamid R. Barzegar. Service migration in multi-domain cellular networks based on machine learning approaches. In *2020 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 1–8, 2020.

[71] Amine Abouaomar, Zoubeir Mlika, Abderrahime Filali, Soumaya Cherkaoui, and Abdellatif Kobbane. A deep reinforcement learning approach for service migration in mec-enabled vehicular networks. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pages 273–280, 2021.

[72] Rami Akrem Addad, Diego Leonel Cadette Dutra, Tarik Taleb, and Hannu Flinck. AI-Based Network-Aware Service Function Chain Migration in 5G and Beyond Networks. *IEEE Transactions on Network and Service Management*, 19(1):472–484, 2022.

[73] CRIU - a utility to checkpoint/restore Linux tasks. [Online], `https://criu.org`.

[74] Jakob Schrettenbrunner. *Migrating Pods in Kubernetes*. PhD thesis, 12 2020.

[75] Podmigration-Operator: An operator that supports Pod Migration in K8s. [Online], `https://github.com/SSU-DCN/podmigration-operator`.

[76] Michael Hines and Kartik Gopalan. Post-copy based live virtual machine migration using pre-paging and dynamic self-ballooning. pages 51–60, 01 2009.

[77] UERANSIM: 5G UE and RAN (gNodeB) simulator. [Online], `https://github.com/aligungr/UERANSIM`.

[78] Michael Gundall, Julius Stegmann, Christopher Huber, and Hans Schotten. Towards organic 6g networks: Virtualization and live migration of core network functions. 10 2021.

[79] Satyam Dwivedi, Ritesh Shreevastav, Florent Munier, Johannes Nygren, Iana Siomina, Yazid Lyazidi, Deep Shrestha, Gustav Lindmark, Per Ernström, Erik Stare, Sara M. Razavi, Siva Muruganathan, Gino Masini, Åke Busin, and Fredrik Gunnarsson. Positioning in 5g networks, 2021.

[80] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[81] OOKLA 5G MAP™: The interactive Ookla 5G Map tracks 5G rollouts in cities across the globe. [Online], `https://www.speedtest.net/ookla-5g-map`.

[82] Muhammad Tirmazi, Adam Barker, Nan Deng, Md Ehtesam Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: the next generation. In *EuroSys'20*, Heraklion, Crete, 2020.

[83] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[84] Zhi xiong Xu, Lei Cao, Xi liang Chen, Chenxi Li, Yongliang Zhang, and Jun Lai. Deep reinforcement learning with sarsa and q-learning: A hybrid approach. *IEICE Trans. Inf. Syst.*, 101-D:2315–2322, 2018.

[85] S Rohith Raj, R Rohith, Minal Moharir, and G Shobha. SCAPY- A powerful interactive packet manipulation program. In *2018 International Conference on Networking, Embedded and Wireless Systems (ICNEWS)*, pages 1–5, 2018.

[86] Nei Kato, Bomin Mao, Fengxiao Tang, Yuichi Kawamoto, and Jiajia Liu. Ten Challenges in Advancing Machine Learning Technologies toward 6G. *IEEE Wireless Communications*, 27(3):96–103, 2020.

[87] Ioannis Tomkos, Dimitrios Klonidis, Evangelos Pikasis, and Sergios Theodoridis. Toward the 6G Network Era: Opportunities and Challenges. *IT Professional*, 22(1):34–38, 2020.

[88] Chia-Yu Chang, Navid Nikaein, Osama Arouk, Kostas Katsalis, Adlen Ksentini, Thierry Turletti, and Konstantinos Samdanis. Slice Orchestration for Multi-Service Disaggregated Ultra-Dense RANs. *IEEE Communications Magazine*, 56(8):70–77, 2018.

[89] Chang Ge, Ning Wang, Severin Skillman, Gerry Foster, and Yue Cao. QoE-Driven DASH Video Caching and Adaptation at 5G Mobile Edge. In *ACM Conference on Information-Centric Networking*, ACM-ICN '16, page 237–242, New York, NY, USA, 2016. Association for Computing Machinery.

[90] Rob Smith, Connor Freeberg, Travis Machacek, and Venkatesh Ramaswamy. An O-RAN Approach to Spectrum Sharing Between Commercial 5G and Government Satellite Systems. In *IEEE Military Communications Conference (MILCOM)*, pages 739–744, 2021.

[91] Line MP Larsen, Aleksandra Checko, and Henrik L Christiansen. A survey of the functional splits proposed for 5G mobile crosshaul networks. *IEEE Communications Surveys & Tutorials*, 21(1):146–172, 2018.

[92] Andres Garcia-Saavedra and Xavier Costa-Perez. O-RAN: Disrupting the virtualized RAN ecosystem. *IEEE Communications Standards Magazine*, 2021.

[93] Amitava Ghosh, Andreas MÃ¤der, Matthew Baker, and Devaki Chandramouli. 5G Evolution: A View on 5G Cellular Technology Beyond 3GPP Release 15. *IEEE Access*, PP:1–1, 09 2019.

[94] Richard Cziva, Christos Anagnostopoulos, and Dimitrios P Pezaros. Dynamic, latency-optimal vNF placement at the network edge. In *IEEE conference on computer communications (INFOCOM)*, pages 693–701. IEEE, 2018.

[95] Zhichao Xu, Xiaoning Zhang, Shui Yu, and Ji Zhang. Energy-Efficient Virtual Network Function Placement in Telecom Networks. In *International Conference on Communications (ICC)*, pages 1–7, 2018.

[96] Ioannis Sarrigiannis, Kostas Ramantas, Elli Kartsakli, Prodromos-Vasileios Mekikis, Angelos Antonopoulos, and Christos Verikoukis. Online VNF Lifecycle Management in an MEC-Enabled 5G IoT Architecture. *IEEE Internet of Things Journal*, 7(5):4183–4194, 2020.

[97] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. Towards load-balanced VNF assignment in geo-distributed NFV Infrastructure. In *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, pages 1–10, 2017.

[98] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. Adaptive VNF scaling and flow routing with proactive demand prediction. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 486–494. IEEE, 2018.

[99] Dejene Boru Oljira, Karl-Johan Grinnemo, Javid Taheri, and Anna Brunstrom. A model for QoS-aware VNF placement and provisioning. In *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–7, 2017.

[100] Long Qu, Chadi Assi, and Khaled Shaban. Delay-Aware Scheduling and Resource Optimization With Network Function Virtualization. *IEEE Transactions on Communications*, 64(9):3746–3758, 2016.

[101] Dinesh Kumar, Somnath Chakrabarti, Ashok Sunder Rajan, and Jim Huang. Scaling Telecom Core Network Functions in Public Cloud Infrastructure. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 9–16. IEEE, 2020.

[102] Imad Alawe, Adlen Ksentini, Yassine Hadjadj-Aoul, and Philippe Bertin. Improving Traffic Forecasting for 5G Core Network Scalability: A Machine Learning Approach. *IEEE Network*, 32(6):42–49, 2018.

[103] Imad Alawe, Yassine Hadjadj-Aoul, Adlen Ksentini, Philippe Bertin, and Davy Darche. On the scalability of 5G core network: The AMF case. In *IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 1–6, 2018.

[104] ITRI. Athena Orchestrator - O-RAN SMO & RIC, note=[Online], `https://event.itri.org/CES2023/tech_details/22`.

[105] Chieh-Chun Chen, Chia-Yu Chang, and Navid Nikaein. FlexSlice: Flexible and real-time programmable RAN slicing framework. In *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, pages 3807–3812, 2023.

[106] Jasneet Kaur, M Arif Khan, Mohsin Iftikhar, Muhammad Imran, and Qazi Emad Ul Haq. Machine Learning techniques for 5G and beyond. *IEEE Access*, 9:23472–23488, 2021.

[107] Ons Aouedi, Kandaraj Piamrat, Salima Hamma, and J. Perera. Network traffic analysis using machine learning: an unsupervised approach to understand and slice your network. *annals of telecommunications*, 11 2021.

[108] Qiaofeng Qin, Konstantinos Poularakis, Kin K. Leung, and Leandros Tassiulas. Line-speed and scalable intrusion detection at the network edge via federated learning. In *IFIP Networking Conference (Networking)*, pages 352–360, 2020.

[109] Jun Zhang, Yang Xiang, Yu Wang, Wanlei Zhou, Yong Xiang, and Yong Guan. Network traffic classification using correlation information. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):104–117, 2013.

[110] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*, MineNet '06, page 281–286, New York, NY, USA, 2006. Association for Computing Machinery.

[111] Michael Finsterbusch, Chris Richter, Eduardo Rocha, Jean-Alexander Muller, and Klaus Hanssgen. A survey of payload-based traffic classification approaches. *IEEE Communications Surveys & Tutorials*, 16(2):1135–1156, 2014.

[112] Thuy T.T. Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008.

[113] Shahbaz Rezaei and Xin Liu. Deep learning for encrypted traffic classification: An overview. *IEEE Communications Magazine*, 57(5):76–81, 2019.

[114] Alberto Dainotti, Antonio Pescape, and Kimberly C. Claffy. Issues and future directions in traffic classification. *IEEE Network*, 26(1):35–40, 2012.

[115] Chunxiao Jiang, Haijun Zhang, Yong Ren, Zhu Han, Kwang-Cheng Chen, and Lajos Hanzo. Machine Learning paradigms for Next-Generation Wireless Networks. *IEEE Wireless Communications*, 24(2):98–105, 2016.

[116] Marcin Dryjański, Łukasz Kułacz, and Adrian Kliks. Toward Modular and Flexible Open RAN Implementations in 6G Networks: Traffic Steering Use Case and O-RAN xApps. *Sensors*, 21(24), 2021.

[117] Anurag Thantharate, Ankita Vijay Tondwalkar, Cory Beard, and Andres Kwasinski. Eco6g: Energy and cost analysis for network slicing deployment in beyond 5g networks. *Sensors*, 22(22), 2022.

[118] Bouziane Brik and Adlen Ksentini. On Predicting Service-oriented Network Slices Performances in 5G: A Federated Learning Approach. In *IEEE Conference on Local Computer Networks (LCN)*, pages 164–171. IEEE, 2020.

[119] Chia-Yu Chang and Navid Nikaein. Closing in on 5G control apps: enabling multiservice programmability in a disaggregated radio access network. *IEEE Vehicular Technology Magazine*, 13(4):80–93, 2018.

[120] Qiang Liu, Nakjung Choi, and Tao Han. OnSlicing: Online End-to-End Network Slicing with Reinforcement Learning. In *International Conference on Emerging Networking EXperiments and Technologies (CONEXT*, CoNEXT '21, page 141–153, New York, NY, USA, 2021. Association for Computing Machinery.

[121] Sharvari Ravindran, Saptarshi Chaudhuri, Jyotsna Bapat, and Debabrata Das. Novel adaptive multi-user multi-services scheduling to enhance throughput in 5g-advanced and beyond. *IEEE Transactions on Network and Service Management*, pages 1–1, 2024.

[122] Nazih Salhab, Rami Langar, and Rana Rahim. 5G network slices resource orchestration using Machine Learning techniques. *Computer Networks*, 188:107829, 2021.

[123] Ilias Chatzistefanidis, Nikos Makris, Virgilios Passas, and Thanasis Korakis. UE Statistics Time-Series (CQI) in LTE Networks, 2022.

[124] Theodoros Tsourdinis, Ilias Chatzistefanidis, Nikos Makris, and Thanasis Korakis. Ue network traffic time-series (applications, throughput, latency, cqi) in lte/5g networks, 2022.

[125] Dor Green. Pyshark: Python wrapper for tshark, allowing python packet parsing using wireshark dissectors. [Online], `https://github.com/KimiNewt/pyshark`.

[126] Chen Chen, Wei Wang, and Bo Li. Round-Robin Synchronization: Mitigating Communication Bottlenecks in Parameter Servers. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 532–540, 2019.

[127] Raja Ettiane, Abdelaali Chaoub, and Rachid Elkouch. Toward securing the control plane of 5G mobile networks against DoS threats: Attack scenarios and promising solutions. *Journal of Information Security and Applications*, 61:102943, 2021.

[128] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges, 2022.

[129] Song Wang, Juan Fernando Balarezo Serrano, Kandeepan Sithamparanathan, Akram Al-Hourani, Karina Gomez Chavez, and Ben Rubinstein. Machine Learning in Network Anomaly Detection: A Survey. *IEEE Access*, PP:1–1, 11 2021.

[130] Lorenzo Fernández Maimó, Ángel Luis Perales Gómez, Félix J. García Clemente, Manuel Gil Pérez, and Gregorio Martínez Pérez. A Self-Adaptive Deep Learning-Based System for Anomaly Detection in 5G Networks. *IEEE Access*, 6:7700–7712, 2018.

[131] Jordan Lam and Robert Abbas. Machine Learning based Anomaly Detection for 5G Networks, 2020.

[132] Salah Bin Ruba, Nour El-Houda Yellas, and Stefano Secci. Anomaly Detection for 5G Softwarized Infrastructures with Federated Learning. In *2022 1st International Conference on 6G Networking (6GNet)*, pages 1–4, 2022.

[133] Jung-Erh Chang, Yi-Chen Chiu, Yi-Wei Ma, Zhi-Xiang Li, and Cheng-Long Shao. Packet Continuity DDoS Attack Detection for Open Fronthaul in ORAN System. In

NOMS 2024-2024 IEEE Network Operations and Management Symposium, pages 1–5, 2024.

[134] C.T. Shen, Y.Y. Xiao, Y.W. Ma, J.L. Chen, Cheng-Mou Chiang, S.J. Chen, and Y. C. Pan. Security Threat Analysis and Treatment Strategy for ORAN. In *2022 24th International Conference on Advanced Communication Technology (ICACT)*, pages 417–422, 2022.

[135] Giorgi Iashvili, Maksim Iavich, Razvan Bocu, Roman Odarchenko, and Sergiy Gnatyuk. Intrusion Detection System for 5G with a Focus on DOS/DDOS Attacks. In *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 2, pages 861–864, 2021.

[136] Bruno Missi Xavier, Merim Dzaferagic, Diarmuid Collins, Giovanni Comarela, Magnos Martinello, and Marco Ruffini. Machine Learning-based Early Attack Detection Using Open RAN Intelligent Controller, 2023.

[137] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D. Sutton, Pablo Serrano, Cristina Cano, and Doug J. Leith. srsLTE: an open-source platform for LTE evolution and experimentation. In *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, WiNTECH '16, page 25–32, New York, NY, USA, 2016. Association for Computing Machinery.

[138] Azuka Chiejina, Brian Kim, Kaushik Chowhdury, and Vijay K. Shah. System-level Analysis of Adversarial Attacks and Defenses on Intelligence in O-RAN based Cellular Networks, 2024.

[139] F. Bimo, F. Feliana, S. Liao, C. Lin, D. F. Kinsey, J. Li, R. Jana, R. Wright, and R. Cheng. OSC Community Lab: The Integration Test Bed for O-RAN Software Community. In *2022 IEEE Future Networks World Forum (FNWF)*, pages 513–518, Los Alamitos, CA, USA, oct 2022. IEEE Computer Society.

[140] Raoul Raftopoulos, Salvatore d'oro, Tommaso Melodia, and Giovanni Schembra. DRL-based Latency-Aware Network Slicing in O-RAN with Time-Varying SLAs. In

Proceedings of the International Conference on Communications (ICNC) 2024, 02 2024.

[141] Leonardo Bonati, Michele Polese, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. OpenRAN Gym: An Open Toolbox for Data Collection and Experimentation with AI in O-RAN, 2022.

[142] Theodoros Tsourdinis, Ilias Chatzistefanidis, Nikos Makris, Thanasis Korakis, Navid Nikaein, and Serge Fdida. Service-aware real-time slicing for virtualized beyond 5G networks. *Computer Networks*, 247:110445, 2024.

[143] Asheesh Tripathi, Jaswanth S R Mallu, Md. Habibur Rahman, Abida Sultana, Aditya Sathish, Alexandre Huff, Mayukh Roy Chowdhury, and Aloizio Pereira Da Silva. End-to-End O-RAN Control-Loop For Radio Resource Allocation in SDR-Based 5G Network. In *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, pages 253–254, 2023.

[144] Karim Boutiba, Miloud Bagaa, and Adlen Ksentini. On enabling 5G Dynamic TDD by leveraging Deep Reinforcement Learning and O-RAN. In *2023 IEEE/IFIP NOMS*, 2023.

[145] Serge Fdida, Nikos Makris, Thanasis Korakis, Raffaele Bruno, Andrea Passarella, Panayiotis Andreou, Bartosz Belter, CÃ©dric Crettaz, Walid Dabbous, Yuri Demchenko, and Raymond Knopp. SLICES, a scientific instrument for the networking community. *Computer Communications*, 193:189–203, 2022.

[146] Ibrahim Obeidat, Nabhan Hamadneh, Mouhammd Al-kasassbeh, and Mohammad Almseidin. Intensive Preprocessing of KDD Cup 99 for Network Intrusion Classification Using Machine Learning Techniques, 2018.

[147] Rohith Raj S, Rohith R, Minal Moharir, and Shobha G. SCAPY- A powerful interactive packet manipulation program. In *2018 International Conference on Networking, Embedded and Wireless Systems (ICNEWS)*, pages 1–5, 2018.