UNIVERSITY OF THESSALY

DOCTORAL THESIS

Intelligent Orchestration in Heterogeneous Networked Systems

Author:
Ilias Syrigos

Supervisor:
Prof. Athanasios Korakis

Dissertation Committee:
Prof. Athanasios Korakis
Prof. Leandros Tassiulas
Prof. Antonios Argyriou

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Department of Electrical and Computer Engineering

July 25, 2025

Declaration of Authorship

I, Ilias Syrigos, declare that this thesis titled, "Intelligent Orchestration in Heterogeneous Networked Systems" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"It does not matter how slowly you go as long as you do not stop."

Confucius

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Περίληψη

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Ευφυής Ενορχήστρωση σε Ετερογενή Δικτυωμένα Συστήματα

Ηλίας Συρίγος

Η αυξανόμενη πολυπλοκότητα, ετερογένεια και κλίμακα των σύγχρονων ψηφιακών υποδομών έχει αναδιαμορφώσει τα λειτουργικά όρια των δικτυακών και υπολογιστικών συστημάτων. Περιβάλλοντα που εκτείνονται από πυκνές αστικές ασύρματες αναπτύξεις και κινητά ad hoc δίκτυα έως αποσυσχετισμένα (disaggregated) υπολογιστικά κέντρα δεδομένων και την κατανεμημένη υποδομή του cloud-to-things continuum απαιτούν πλέον ευφυείς, αποδοτικούς και ανθεκτικούς μηχανισμούς ενορχήστρωσης. Τα συστήματα αυτά πρέπει να προσαρμόζονται δυναμικά σε πραγματικό χρόνο, να διαχειρίζονται περιορισμένους και ευμετάβλητους πόρους και να παραμένουν ενεργειακά αποδοτικά, ενώ ταυτόχρονα ικανοποιούν αυστηρές απαιτήσεις απόδοσης.

Η κεντρική υπόθεση αυτής της διατριβής είναι ότι η ευφυής ενορχήστρωση σε τέτοια περιβάλλοντα καθίσταται εφικτή μέσω του συνδυασμού διαστρωματικής παρατηρησιμότητας, λογισμικά καθοριζόμενου ελέγχου και αυτοματοποίησης βασισμένης στη μηχανική μάθηση. Αντί για μια ενιαία μονολιθική λύση, η διατριβή προτείνει μια σειρά από εξειδικευμένα πλαίσια ενορχήστρωσης, σχεδιασμένα με βάση τις αρχιτεκτονικές και λειτουργικές ιδιαιτερότητες των εκάστοτε περιβαλλόντων. Όλα βασίζονται στην ίδια θεμελιώδη αρχή: η ενορχήστρωση πρέπει να είναι απολύτως εξαρτημένη από το περιβάλλον, καθοδηγούμενη από δεδομένα και προσαρμοζόμενη δυναμικά.

Αρχικά εξετάζουμε τα ασύρματα τοπικά δίκτυα (WLANs), στα οποία η συμφόρηση, οι παρεμβολές και το φαινόμενο των «κρυφών κόμβων» οδηγούν σε επίμονη υποβάθμιση της απόδοσης. Αναπτύσσουμε και υλοποιούμε ένα σύστημα διάγνωσης βασισμένο στο επίπεδο MAC, το οποίο αξιοποιεί στατιστικά στοιχεία που παρέχονται από εμπορικά access points για να εντοπίσει και να ταξινομήσει τις παθολογίες του WLAN. Το πλαίσιο αυτό, χωρίς να απαιτεί καμία υλική τροποποίηση, λειτουργεί τόσο σε ενεργή όσο και σε παθητική λειτουργία, προσφέροντας υψηλή ακρίβεια και άμεσες δυνατότητες εφαρμογής.

Στη συνέχεια εστιάζουμε σε κινητά ad hoc δίκτυα (MANETs), όπου η συγκεντρωτική ενορχήστρωση εμφανίζεται ασύμβατη με τη δυναμική τοπολογία. Για την αντιμετώπιση του προβλήματος, σχεδιάζουμε ένα ανθεκτικό πλαίσιο Software Defined

Networking (SDN), το οποίο υποστηρίζει κατανεμημένη εκλογή ελεγκτών, in-band σηματοδοσία και προσαρμοστική δρομολόγηση βάσει διαστρωματικών δεικτών ποιότητας σύνδεσης. Επεκτείνουμε το σύστημα αυτό με πράκτορες ενισχυτικής μάθησης (reinforcement learning) που ρυθμίζουν αυτόνομα τις παραμέτρους του καναλιού ελέγχου (control plane), διατηρώντας τη σταθερότητα του δικτύου σε συνθήκες κινητικότητας και μεταβαλλόμενων δικτυακών συνθηκών. Τα πειραματικά αποτελέσματα δείχνουν σημαντικές βελτιώσεις σε ρυθμοαπόδοση, καθυστέρηση και ανθεκτικότητα.

Στον χώρο των datacenters, διερευνούμε τις προκλήσεις ενορχήστρωσης σε αποσυσχετισμένα συστήματα, όπου οι πόροι υπολογισμού και μνήμης είναι διαχωρισμένοι και διασυνδέονται μέσω προγραμματιζόμενων οπτικών υφιστάμενων δικτύων. Σχεδιάζουμε ένα επίπεδο ενορχήστρωσης βασισμένο σε γράφους, τον Software Defined Memory (SDM) controller, που επιτρέπει αποδοτικό προγραμματισμό, δυναμική σύνδεση απομακρυσμένων μνημών και ταχεία αναδιαμόρφωση της τοπολογίας. Η υλοποίηση, σε ενοποίηση με το OpenStack, επιδεικνύει σημαντικές βελτιώσεις στον χρόνο προγραμματισμού και στην ενεργειακή αποδοτικότητα.

Επιπρόσθετα, εξετάζουμε το ζήτημα της ομοιογενούς και ασφαλούς ενορχήστρωσης πειραματικών, cloud-native (5G) υποδομών που ανήκουν σε διαφορετικούς διαχειριστικούς τομείς και γεωγραφικές περιοχές. Προτείνουμε ένα ολοκληρωμένο πλαίσιο πολυσυσταδικής (multi-cluster) ενορχήστρωσης, το οποίο αξιοποιεί προηγμένες τεχνολογίες ανοιχτού κώδικα για τη διαχείριση των συστάδων και την ασφαλή εξαγωγή υπηρεσιών μεταξύ απομακρυσμένων διοικητικών domains. Το πλαίσιο αυτό αναπτύσσεται πάνω από δικτυακή υποδομή υψηλής ταχύτητας και αξιολογείται σε πραγματικά πειραματικά περιβάλλοντα, αποδεικνύοντας την ικανότητά του να υποστηρίζει cloud-native 5G φορτία με αποδοτικότητα και διαφάνεια.

Τέλος, στο πλαίσιο της υποδομής υπολογιστικού συνεχούς (computing continuum), εξετάζουμε τις λειτουργίες MLOps και την ανάπτυξη ροών εργασίας μηχανικής μάθησης σε ετερογενείς πόρους cloud, edge και far-edge. Παρουσιάζουμε αρχικά ένα πλαίσιο ενορχήστρωσης που αποσυνθέτει τα ML pipelines σε λειτουργικές ενότητες και τις προγραμματίζει σύμφωνα με τις απαιτήσεις τους. Επιπλέον, σχεδιάζουμε το EELAS, έναν ενεργειακά αποδοτικό και χρονικά ευαίσθητο προγραμματιστή (scheduler) για εργασίες πρόβλεψης (inference), ενσωματωμένο στο Kubernetes και επικυρωμένο σε πραγματικές δοκιμές, επιτυγχάνοντας σημαντική μείωση στην κατανάλωση ενέργειας ενώ διατηρείται η καθορισμένη καθυστέρηση στην εκτέλεση.

Συνολικά, οι συνεισφορές αυτές παρέχουν μια πλήρη και ρεαλιστική προσέγγιση στην ενορχήστρωση συστημάτων με κρίσιμες απαιτήσεις απόδοσης και ενεργειακής αποδοτικότητας σε ετερογενή δικτυακά περιβάλλοντα. Αποδεικνύεται ότι οι αρχές της διαστρωματικής σχεδίασης, της προγραμματισιμότητας και της προσαρμογής μέσω

μάθησης μπορούν να εφαρμοστούν αποτελεσματικά για την αντιμετώπιση των προκλήσεων συντονισμού των υποδομών επόμενης γενιάς.

UNIVERSITY OF THESSALY

Abstract

Department of Electrical and Computer Engineering

Doctor of Philosophy

"Intelligent Orchestration in Heterogeneous Networked Systems"

by Ilias Syrigos

The increasing complexity, heterogeneity, and scale of modern digital infrastructures have redefined the operational boundaries of networking and computing systems. Environments spanning dense urban wireless deployments, mobile ad hoc networks, disaggregated cloud datacenters, federated 5G testbeds, and the distributed cloud-to-things continuum now demand intelligent, efficient, and resilient orchestration mechanisms. These systems must adapt in real time, manage constrained and fluctuating resources, and remain energy-aware, while still delivering performance guarantees.

The central hypothesis of this thesis is that intelligent orchestration in such environments is feasible through a combination of cross-layer observability, software-defined control, and learning-based automation. Rather than proposing a single monolithic solution, this thesis introduces several specialized orchestration frameworks that are tailored to the architectural and operational constraints and needs of their respective environments. These frameworks are based on the same foundational concept: orchestration must be context-aware, data-driven, and dynamically adaptive.

We begin with wireless local area networks, where contention, interference, and hidden terminal effects lead to persistent performance degradation. We design and implement a MAC-layer-based diagnosis system that utilizes driver-exposed statistics from commercial access points to identify and classify WLAN pathologies. Without requiring any hardware modifications, this lightweight framework operates in both active and passive modes, achieving high accuracy and offering practical deployment potential.

Next, we turn to mobile ad hoc networks (MANETs), where centralized orchestration appears incompatible with volatile topologies. To address this, we design a fault-tolerant Software Defined Networking (SDN) framework that supports distributed controller election, in-band signaling, and adaptive routing based on cross-layer link quality metrics. We further extend this system with reinforcement learning agents that dynamically tune SDN control-plane parameters to maintain network stability under mobility and channel fluctuations. Our experiments show significant improvements in throughput, latency, and resilience.

In the datacenter domain, we investigate orchestration challenges in disaggregated systems, where compute and memory are decoupled and interconnected over programmable optical fabrics. We design a graph-based orchestration plane, the Software Defined Memory (SDM) controller, that enables efficient scheduling, dynamic

attachment of memory resources, and fast topology reconfiguration. Our implementation, integrated with OpenStack, demonstrates substantial gains in scheduling latency and energy efficiency.

Building upon the growing need for distributed experimentation across cloud-native, federated (5G) testbeds, we propose a multi-cluster orchestration framework that leverages state-of-the-art open-source solutions to securely expose services and establish inter-cluster connectivity across geographically dispersed administrative domains. Deployed over a high-speed fabric, the system enables cloud-native 5G workloads to be orchestrated transparently across distinct infrastructures, while preserving isolation and maintaining performance guarantees.

Shifting to the computing continuum, we explore machine learning operations (MLOps) and the deployment of AI workflows across heterogeneous cloud, edge, and far-edge resources. First, we introduce an orchestration framework that decomposes ML pipelines into modular components and schedules them according to their resource and latency requirements. Then, we design EELAS, an energy-efficient and latency-aware scheduler for ML inference workloads that integrates into Kubernetes and is validated in real-world testbeds, achieving notable reductions in energy consumption while meeting application-level latency constraints.

In overall, these contributions offer a comprehensive and realistic approach to orchestrating performance- and energy-critical systems in heterogeneous networking environments. They demonstrate that the principles of cross-layer design, programmability, and learning-based adaptation can be effectively applied to address the coordination challenges posed by next-generation infrastructures.

Acknowledgements

This dissertation marks the culmination of many years of study and research at the Department of Electrical and Computer Engineering of the University of Thessaly. Along this journey, I have been fortunate to receive support, guidance, and encouragement from many people, without whom this work would not have been possible.

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Thanasis Korakis, for giving me the opportunity to join his lab and for providing me with an environment where I could learn, explore, and grow. His continuous guidance and mentoring, not only on a professional but also on a personal level, have been invaluable throughout my journey. I am especially thankful for the trust he placed in me and for his unwavering belief in my capabilities, which gave me the confidence to overcome challenges and pursue my research with determination.

I would also like to sincerely thank my co-advisors, Prof. Leandros Tassiulas and Prof. Antonios Argyriou, as well as the members of my dissertation committee, Prof. Iordanis Koutsopoulos, Prof. Spyridon Lalis, Prof. Christos Antonopoulos, and Assist. Prof. Paris Flegkas, for honoring me with their participation in the committee and for offering their time and expertise.

Next, I would like to thank all the colleagues, collaborators, and lab-mates with whom I had the opportunity to work throughout my PhD years. Their contributions, support, and teamwork have been essential in shaping my research. I am especially grateful to Stratos Keranidis, with whom I worked during my early years in the lab. From the very first steps of my research, he has been a mentor, a source of motivation, and a good friend whose guidance has been very important to me. I also wish to express my appreciation to my close collaborators during my PhD: Nikos Makris, Dimitris Syrivelis, Kostas Choumas, Apostolis Prassas, and Ippokratis Koukoulis, as well as to the colleagues with whom I worked closely on the various research projects undertaken by our lab, such as Apostolos Apostolaras, whose contributions and teamwork were invaluable in advancing our joint efforts. I am also thankful to all other members, current and former, of NITLab for their collaboration, stimulating discussions, and for creating an enjoyable work environment, with special thanks to Virgilios Passas, Kostas Chounos, Christina Madelou, Donatos Stavropoulos, and Giannis Kazdaridis, whose friendship, support, patience, and the fun moments we shared made this journey truly memorable.

Above all, I am deeply grateful to my parents and my brother for their love, endless support, and strong belief in me. I dedicate this work to them with profound gratitude.

Publications

The ideas as well as the algorithms, figures and results of this thesis, are part of the following publications:

- **Journals**

  [J1:] I. Syrigos, A. Prassas, I. Koukoulis, K. Choumas, and T. Korakis. **"Dynamic SDN Configuration in MANETs: A Reinforcement Learning Approach"**, *(submitted for publication) to IEEE Access, 2025, IEEE*

- **Conferences**

  [C1:] I. Syrigos, N. Makris, and T. Korakis **"Multi-Cluster Orchestration of 5G Experimental Deployments in Kubernetes over High-Speed Fabric."**, *in the proceedings of IEEE Globecom Workshops (GC Wkshps), 2023, IEEE*

  [C2:] I. Syrigos, D. Kefalas, N. Makris, and T. Korakis **"EELAS: Energy Efficient and Latency Aware Scheduling of Cloud-Native ML Workloads"**, *in the proceedings of International Conference on COMmunication Systems & NETworkS (COMSNETS), 2023, IEEE*

  [C3:] I. Syrigos, N. Angelopoulos, and T. Korakis **"Optimization of Execution for Machine Learning Applications in the Computing Continuum"**, *in the proceedings of IEEE Conference on Standards for Communications and Networking (CSCN), 2022, IEEE*

  [C4:] I. Syrigos, D. Syrivelis, and T. Korakis **"On the Implementation of a Software-Defined Memory Control Plane for Disaggregated Datacenters"**, *in the proceedings of IEEE International Conference on Cloud Networking (CloudNet), 2022, IEEE*

  [C5:] I. Syrigos, N. Sakellariou, S. Keranidis, and T. Korakis **"On the employment of machine learning techniques for troubleshooting WiFi networks"**, *in the proceedings of IEEE Annual Consumer Communications & Networking Conference (CCNC), 2019, IEEE / CCNC*

  [C6:] I. Syrigos, S. Keranidis, T. Korakis, and C. Dovrolis **"Enabling Wireless LAN Troubleshooting"**, *in the proceedings of International Conference on Passive and Active Network Measurement (PAM), 2015, Springer*

Furthermore, there were research efforts within the same or previous periods, led to the following publications which are not directly correlated with this thesis:

- **Journals**

[J1:] K. Choumas, I. Syrigos, T. Korakis, and L. Tassiulas **"Video-aware multicast opportunistic routing over 802.11 two-hop mesh networks"**, *IEEE Transactions on Vehicular Technology*, 2017, IEEE

[J2:] I. Koukoulis, I. Syrigos, A. Prassas, K. Choumas and T. Korakis **"Routing Optimization Under an SDN Architecture for 802.11 MANETs"**, *(accepted for publication) in IEEE Transactions on Network and Service Management*, 2025, IEEE

- **Book Chapters**

[B1:] N. Alachiotis, A. Andronikakis, O. Papadakis, D. Theodoropoulos, D. Pnevmatikatos, D. Syrivelis, A. Reale, K. Katrinis, G. Zervas, V. Mishra, H. Yuan, I. Syrigos, I. Igoumenos, T. Korakis, M. Torrents, and F. Zyulkyarov **"dReDBox: A disaggregated architectural perspective for data centers"**, *Hardware Accelerators in Data Centers, 2019, Springer*

- **Conferences**

[C1:] I. Koukoulis, I. Syrigos, and T. Korakis **"Self-Supervised Transformer-based Contrastive Learning for Intrusion Detection Systems"**, *in the proceedings of 2025 IFIP Networking Conference (IFIP Networking), 2025, IEEE*

[C2:] K. Kyriakou, A. Apostolaras, P. Velentzas, I. Syrigos, S. Maglavera, S. Evangelatos, E. Veroni, and T. Korakis **"Biometrics Data Space: Ensuring Trustworthy and Secure Data Exchange for Suspect Identification"**, *in the proceedings of 5th International Conference in Electronic Engineering, Information Technology & Education (EEITE), 2024, IEEE*

[C3:] D. Kefalas, S. Christakis, S. Fdida, N. Makris, I. Syrigos, V. Passas, and T. Korakis **"slAIces: an LLM Chatbot for Simplifying Experiments with the SLICES-RI."**, *in the proceedings of 2024 IFIP Networking Conference (IFIP Networking), 2024, IEEE*

[C4:] I. Syrigos, I. Koukoulis, A. Prassas, K. Choumas, and T. Korakis **"On the Implementation of a Cross-Layer SDN Architecture for 802.11 MANETs"**, *in the proceedings of IEEE International Conference on Communications (ICC), 2023, IEEE*

[C5:] M. Bielski, I. Syrigos, K. Katrinis, D. Syrivelis, A. Reale, D. Theodoropoulos, N. Alachiotis, D. Pnevmatikatos, E. Pap, G. Zervas, V. Mishra, A. Saljoghei, A. Rigo, J. Zazo, S. Lopez-Buedo, M. Torrents, F. Zyulkyarov, M. Enrico, and O. Gonzalez De Dios **"dReDBox: Materializing a full-stack rack-scale system prototype of a next-generation disaggregated datacenter"**, *in the proceedings of Design, Automation & Test in Europe Conference & Exhibition*

(DATE), 2018, IEEE

[C6:] D. Syrivelis, A. Reale, K. Katrinis, I. Syrigos, M. Bielski, D. Theodoropoulos, D. Pnevmatikatos, and G. Zervas **"A software-defined architecture and prototype for disaggregated memory rack scale systems"**, *in the proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2017, IEEE*

[C7:] A. Apostolaras, K. Choumas, I. Syrigos, I. Koutsopoulos, T. Korakis, A. Argyriou, and L. Tassiulas **"On the implementation of relay selection strategies for a cooperative diamond network"**, *in the proceedings of the IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), 2013, IEEE*

- **Demos**

[D1:] M. Enrico, V. Mishra, A. Saljoghei, M. Bielski, E. Pap, I. Syrigos, O. Gonzalez De Dios, D. Theodoropoulos, D. Pnevmatikatos, A. Reale, D. Syrivelis, G. Zervas, N. Parsons, and K. Katrinis **"Demonstration of NFV for mobile edge computing on an optically disaggregated datacentre in a box"**, *in the proceedings of Optical Fiber Communications Conference and Exposition (OFC), 2018, IEEE*

[D2:] A. Saljoghei, V. Mishra, M. Bielski, I. Syrigos, K. Katrinis, D. Syrivelis, A. Reale, D. Pnevmatikatos, D. Theodoropoulos, M. Enrico, N. Parsons, and G. Zervas **"dreddbox: Demonstrating disaggregated memory in an optical data centre"**, *in the proceedings of Optical Fiber Communications Conference and Exposition (OFC), 2018, IEEE*

[D3:] I. Syrigos, K. Choumas, T. Korakis, and L. Tassiulas, **"Demonstration of a Video-aware Multicast Opportunistic Routing protocol over 802.11 two-hop mesh networks"**, *in the proceedings of IEEE International Conference on Sensing, Communication, and Networking (SECON), 2014, IEEE*

[D4:] A. Apostolaras, K. Choumas, I. Syrigos, G. Kazdaridis, T. Korakis, I. Koutsopoulos, A. Argyriou, and L. Tassiulas, **"A demonstration of a relaying selection scheme for maximizing a diamond network's throughput"**, *in the proceedings of he EAI International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom), 2012, EAI*

Contents

Declaration of Authorship **iii**

Abstract **ix**

Acknowledgements **xi**

Publications **xiii**

1 Introduction **1**

1.1 Motivation - Problem Statement 1

1.2 Thesis Synopsis - Contributions 3

2 Enabling Wireless LAN Troubleshooting using MAC-layer Metrics **7**

2.1 Introduction 7

2.2 Related Work 8

2.3 MAC-layer Statistics 9

2.4 IEEE 802.11 Related Pathologies 9

2.4.1 Medium Contention 10

2.4.2 Frame Loss 10

2.5 Detection Methodology 11

2.6 Experimentation with Proposed Metrics 12

2.6.1 Contention with 802.11 terminals 13

2.6.2 Contention with non-802.11 devices 14

2.6.3 Low SNR 14

2.6.4 Hidden Terminal 15

2.6.5 Capture Effect 16

2.6.6 Framework Enhancement and Results Summary 16

2.7 Framework Evaluation 17

2.7.1 Contention with 802.11 terminals 17

2.7.2 Frame Loss . . . 18
2.8 Conclusions and Future Work . . . 18

3 On the Employment of Machine Learning Techniques for Troubleshooting WiFi Networks 21

3.1 Introduction . . . 21
3.2 Related Work . . . 23
3.3 Methodology . . . 23
3.3.1 Sampled metrics . . . 24
3.3.2 Active probing . . . 24
3.3.3 Passive monitoring . . . 25
3.3.4 Experimentation in controlled environment . . . 25
3.4 Evaluation of classification algorithms . . . 28
3.4.1 Classification Algorithms . . . 28
3.4.2 Data Modeling and Feature Extraction . . . 29
3.4.3 Classifiers Implementation . . . 29
3.4.4 Pathology Detection Performance . . . 31
3.5 Conclusion . . . 31

4 On the Implementation of a Cross-Layer SDN Architecture for 802.11 MANETs 33

4.1 Introduction . . . 33
4.2 Related Work . . . 35
4.3 Proposed SDN Scheme . . . 36
4.3.1 Overall Design . . . 36
4.3.2 Integration with IEEE 802.11 . . . 37
4.3.3 Fault Tolerance . . . 37
4.3.4 Topology Discovery . . . 38
4.3.5 Flow Entries Management . . . 39
4.3.6 Cross-layer SDN-MAC integration . . . 40
4.4 Evaluation . . . 42
4.4.1 Testbed Setup . . . 42
4.4.2 Contending Link Experiment . . . 43
4.5 Conclusion . . . 45

5 Dynamic SDN Configuration in MANETs: A Reinforcement Learning Approach 47

5.1 Introduction . . . 48

5.2 Related Work . . . 49

5.2.1 SDN-MANET Integration . . . 49

5.2.2 Reinforcement Learning in SDN . . . 50

5.3 System Overview . . . 50

5.3.1 Overall Design . . . 51

5.3.2 Fault Tolerance . . . 51

5.3.3 Topology Discovery . . . 52

5.4 Behavior of System Parameters and Optimization Trade-offs . . . 53

5.4.1 Analysis of System Parameters . . . 53

5.4.2 Experimental Analysis . . . 55

5.5 Self-Adaptive SDN Configuration . . . 59

5.5.1 Overall Architecture . . . 59

5.5.2 Reinforcement Learning Framework . . . 60

5.5.3 Q-Learning . . . 62

5.5.4 Offline Reinforcement Learning . . . 62

5.6 Implementation . . . 64

5.6.1 State and Action Space . . . 64

5.6.2 Reward Function . . . 66

5.6.3 Training . . . 66

5.7 Evaluation . . . 67

5.7.1 Experimental Setup . . . 68

5.7.2 Performance Metrics . . . 68

5.7.3 Results and Discussion . . . 69

5.8 Conclusion . . . 70

6 On the Implementation of a Software-Defined Memory Control Plane for Disaggregated Datacenters 71

6.1 Introduction . . . 72

6.2 Related Work . . . 73

6.3 Overall Hardware Architecture . . . 74

6.4 Software-Defined Memory Control Plane . . . 75

6.5 Graph Model Design . . . 77
6.5.1 Data Schema . . . 77
6.5.2 Database System Implementation . . . 79
6.6 Evaluation . . . 83
6.6.1 Scheduling Time . . . 83
6.6.2 Scheduling Efficiency . . . 84
6.7 Conclusion . . . 85

7 Multi-Cluster Orchestration of 5G Experimental Deployments in Kubernetes over High-Speed Fabric 87
7.1 Introduction . . . 87
7.2 Background and Motivation . . . 89
7.2.1 Virtualization and Containerization . . . 89
7.2.2 Network Disaggregation . . . 90
7.2.3 Workload Orchestration . . . 90
7.3 Implementation Design . . . 91
7.3.1 Overall Architecture . . . 91
7.3.2 Multi-domain Orchestrator . . . 91
7.3.3 Multi-domain Network Stitching . . . 93
7.4 Evaluation . . . 94
7.5 Conclusion . . . 97

8 Optimization of Execution for Machine Learning Applications in the Computing Continuum 99
8.1 Introduction . . . 100
8.2 Related Work . . . 101
8.3 MLOps Architecture Overview . . . 101
8.3.1 ML Workflow . . . 101
8.3.2 Kubeflow . . . 102
8.3.3 Kubernetes . . . 102
8.3.4 MLOps framework . . . 103
8.4 Scheduling Algorithm . . . 104
8.4.1 Kubernetes Scheduling Algorithm . . . 104
8.4.2 Kubernetes Scheduler Extension . . . 105

8.4.3 Machine Learning Application Scheduler . . . 105
8.5 Evaluation . . . 106
8.5.1 Evaluation Setup . . . 107
8.5.2 ML Application . . . 107
8.5.3 Evaluation Results . . . 108
8.6 Conclusion . . . 109

9 EELAS: Energy Efficient and Latency Aware Scheduling of Cloud-Native ML Workloads 111
9.1 Introduction . . . 111
9.2 Related Work . . . 113
9.3 System Model . . . 114
9.3.1 Problem Definition . . . 114
9.3.2 Problem Formulation . . . 115
9.3.3 Heuristic Approach . . . 117
9.4 Evaluation . . . 118
9.4.1 Experimental Setup . . . 118
9.4.2 Experiment Evaluation . . . 120
9.5 Conclusion . . . 121

10 Conclusions and Future Work 123

Bibliography 127

List of Figures

2.1 Taxonomy of IEEE 802.11 Pathologies 10

2.2 Testbed Topology 12

2.4 NCA and FDR Performance of Medium Contention related Pathologies 13

2.5 NCA and FDR performance under frame loss related pathologies: Low-SNR, Hidden Terminal, and Capture Effect. 15

2.6 Detection Performance across different 802.11 contention scenarios. . . 17

2.7 Detection performance across Frame Loss related pathologies. 18

3.1 Data set. 26

4.1 Overall Architecture 36

4.2 Network Topology 43

4.3 Throughput of flow between A and C. 44

5.1 Architectural design of the in-band SDN framework for MANETs. . . . 51

5.2 Plots for total overhead and associated one-way delay for both Raft algorithm and LLDP protocol for varying network sizes and for average node velocity of 0 m/s. 56

5.3 Plots for Master Controller Availability for varying Raft and LLDP-related parameter values, for varying network sizes and for varying average velocities of nodes. 58

5.4 Integration of RL agent into the SDN-MANET framework. 60

5.5 Interactions between agent and the environment [75]. 61

5.6 Normalized Performance with Static Strategy Variance (15 Configurations). 67

6.1 Hardware Architecture 74

6.2 SDM Controller 75

6.3 Graph representation 78

7.1 The overall under-study architecture: we consider geographically distributed clusters that operate independently. On top, the framework adds a management layer through a central hub, and exposes network functions and services to other clusters. 90

7.2 Rancher architecture [110]. 93

7.3 Submariner network stitching among clusters [113]. 94

7.4 Throughput performance results under different settings. 94

7.5 RTT measurements for the different cable drivers. 96

8.1 ML Workflow 101

8.2 Overall Architecture 104

9.1 The overall EELAS architecture 114

9.2 CPU Utilization per node (Edge/Fog/Cloud) and allocation for the inference workloads 120

List of Tables

2.1 Identified Trends per considered Pathology 17

3.1 Decision Trees sets of hyper-parameters values 27
3.2 Random Forests sets of hyper-parameters values 27
3.3 SVM sets of hyper-parameters values 27
3.4 K-Nearest Neighbors sets of hyper-parameters values 27
3.5 Confusion matrix K-Nearest Neighbors - Active Model 29
3.6 Confusion matrix K-Nearest Neighbors - Passive Model 30
3.7 Evaluation results 31

4.1 Experiment Parameters 42
4.2 Observed Metric Values at t=0s 43
4.3 Observed Metric Values at t=15s 44

5.1 Configurations used in the experiments 57
5.2 Default values of parameters 57
5.3 Configuration values of parameters 65

6.1 Basic query execution time. 80
6.2 Optimized query execution time. 80
6.3 Different flavors of CPU cores and RAM 85

7.1 Configurations used in the experiments 95

8.1 Node RTT Values per Location 107
8.2 Model Training Parameters 109

9.1 Hardware specification 120
9.2 Execution time for each workload on different nodes 121

List of Abbreviations

5G	Fifth Generation Mobile Network
6G	Sixth Generation Mobile Network
AI	Artificial Intelligence
AMPDU	Aggregated MAC Protocol Data Unit
AP	Access Point
BSSID	Basic Service Set Identifier
API	Application Programming Interface
CAPEX	Capital Expenditure
CAGR	Compound Annual Growth Rate
CIDR	Classless Inter-Domain Routing
CNF	Containerized Network Function
CORE	Common Open Research Emulator
CPU	Central Processing Unit
CRI	Container Runtime Interface
CU	Central Unit
CW	Contention Window
DAT	Directional Airtime Metric
DL	Downlink
DNS	Domain Name System
DU	Distributed Unit
EELAS	Energy Efficient Latency-Aware Scheduling
EMANE	Extendable Mobile Ad-hoc Network Emulator
ETSI	European Telecommunications Standards Institute
FDR	Frame Delivery Ratio
GUI	Graphical User Interface
IBSS	Independent Basic Service Set
ILP	Integer Linear Programming
IoT	Internet of Things
IP	Internet Protocol
IPsec	Internet Protocol Security
K8s	Kubernetes
KNN	K-Nearest Neighbors
LC	Local Controller
LLDP	Link Layer Discovery Protocol
MAC	Medium Access Control
MANET	Mobile Ad Hoc Network
MC	Master Controller
MCA	Master Controller Availability
MDP	Markov Decision Process
MIPS	Million Instructions Per Second
ML	Machine Learning
MLAS	Machine Learning Aware Scheduler

MLOps	Machine Learning Operations
MTU	Maximum Transmission Unit
NFV	Network Functions Virtualization
NIC	Network Interface Card
OPEX	Operational Expenditure
OS	Operating System
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
RKE	Rancher Kubernetes Engine
RL	Reinforcement Learning
RU	Radio Unit
SCTP	Stream Control Transmission Protocol
SDK	Software Development Kit
SDM	Software Defined Memory
SDN	Software Defined Networking
SLA	Service Level Agreement
SNR	Signal to Noise Ratio
SVM	Support Vector Machine
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
uRLLC	Ultra-Reliable Low-Latency Communication
VM	Virtual Machine
VNF	Virtualized Network Function
VPN	Virtual Private Network
VXLAN	Virtual Extensible LAN
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Network
WNIC	Wireless Network Interface Card
YAML	YAML Ain't Markup Language

Dedicated to my beloved family and friends…

Chapter 1

Introduction

1.1 Motivation - Problem Statement

In recent years, there has been a rapid evolution of digital infrastructures that has been characterized by the increased device diversity, the transition to software-defined architectures, and the growing need for scalable and energy-efficient deployment of data-driven, resource-intensive workloads. These advances have led to the emergence of highly heterogeneous networked systems that span from cloud datacenters and resource-constrained edge environments to tactical mobile ad hoc networks and dense urban wireless deployments. Academic research and the development of technological solutions have brought significant benefits, such as ubiquitous connectivity, flexible resource allocation, and transparent service execution, to relevant stakeholders, including operators, infrastructure providers, service providers, application developers, and end-users. Nevertheless, each of the individual networking environments poses the same fundamental challenge of efficient monitoring, coordination, and management of performance-critical networking and computing resources in an intelligent and automated manner.

In residential WLANs, for instance, the co-existence of numerous deployments in concentrated space in urban areas leads to significant performance degradation that is related to the inherent pathologies of the 802.11 protocol, such as contention, interference, and the hidden terminal phenomenon. However, end-users are not able to diagnose the causes of such performance drop due to their lack of expertise and the lack of portable and accurate network monitoring tools. Diagnosing such pathologies often requires specialized equipment, custom firmware or intrusive modifications to the access point source code. What is lacking is an intelligent, lightweight, and deployable method that leverages available MAC-layer metrics to infer and differentiate between root causes in real time, without the need for specialized hardware or software support.

In mobile ad hoc networks (MANETs), traditional distributed routing protocols, although able to adapt to the volatile network conditions and the dynamic movement of nodes, do not allow for sophisticated routing decisions and the execution of specific Quality of Service (QoS) policies, as they lack the comprehensive view of the network. Software-defined networking promises centralized control that enables intelligent decision-making, software programmability of traditional network functions, flexibility as policies can be adjusted in response to evolving requirements,

and hardware agnosticity by abstracting the control plane from proprietary hardware. Nevertheless, integrating SDN into MANETs is challenging due to the dynamic topology and frequent link failures that undermine the assumption of centralized control and stable communication paths. Therefore, a robust and fault-tolerant implementation of an SDN framework over MANETs is clearly needed, one that does not rely on static configuration but can self-adapt to the underlying network conditions.

In the datacenter domain, advances in virtualization and containerization have drastically improved resource flexibility. Nevertheless, underutilization is still a concern for traditional server deployments as they remain bound by the constraints of the single-machine architecture. To this end, resource disaggregation promises to increase resource efficiency and energy savings by decoupling compute and memory resources into separate, independent pools that are interconnected through programmable, software-defined networks. However, dynamic allocation of the disaggregated resources and the simultaneous coordination of the programmable network infrastructure pose a significant orchestration challenge.

The rise of microservices and cloud-native deployments has led workloads to span across a wide computing continuum that involves cloud, edge, far-edge, and beyond-edge resources. This shift in workload deployment has introduced new challenges in orchestrating resource-intensive tasks with strict latency requirements, such as machine learning inference, in potentially resource-constrained devices with limited energy budgets and computing capacity. Contemporary orchestration frameworks such as Kubernetes were not designed for such environments, as they lack awareness of energy-related metrics and are not optimized for context-aware and continuum-wide coordination. Beyond single-cluster orchestration, emerging deployment scenarios in 5G and experimental infrastructures introduce the need to coordinate workloads across geographically distributed Kubernetes clusters that are interconnected via high-speed programmable networks. In such environments, services must be placed, scaled, and migrated across clusters based on latency, availability, and topology awareness. Such capabilities are not natively supported by existing orchestration stacks. Furthermore, containerized network functions (CNFs) in 5G testbeds demand precise integration with slicing frameworks, multi-tenant isolation, and inter-cluster ingress routing that add more layers of complexity to the orchestration process. Addressing these challenges requires modular orchestration frameworks that extend Kubernetes functionality to support multi-cluster deployments over federated infrastructures. Moreover, orchestration challenges extend beyond infrastructure to the application layer itself. In Machine Learning Operations (MLOps), ML workflows must be decomposed into several stages (data ingestion, training, inference) that have conflicting requirements. For example, training demands high computing capacities, inference demands minimal latency, and data ingestion demands high throughput. As such, monolithic workflow deployments and manual deployment decisions often fail to efficiently balance these trade-offs, leading to suboptimal performance.

All these diverse environments share a common orchestration gap. Despite the variation in infrastructure, protocols, tools, and deployment context, all domains suffer from the same limitation to observe, interpret, and act in a coordinated and adaptive way. Monitoring, when it exists, is often limited, siloed, or disconnected from higher-level control logic, while orchestration often remains static or manual, unaware of real-time conditions, system constraints, and user requirements.

1.2 Thesis Synopsis - Contributions

The aim of this doctoral thesis is to investigate and offer solutions for the problem of intelligent orchestration and monitoring in heterogeneous networked systems that span a wide spectrum of deployment environments, including urban wireless networks, mobile ad hoc networks (MANETs), disaggregated cloud datacenters and resource-constrained edge devices supporting distributed machine learning workloads. Although the operational context differs in each of the environments, the challenges are similar and involve scalability issues, resource variability, limited observability, and the need for performance-and energy-aware automated coordination of computational and networking resources. The central hypothesis of this thesis is that **Cross-layer visibility, coupled with machine learning-based automation and software-defined control, enables intelligent orchestration strategies that can diagnose, adapt, and optimize networked systems across heterogeneous domains.** Thus, the main goal is not to provide a monolithic platform of orchestration but to develop a suite of specialized orchestration frameworks, each of them tailored to the specific requirements and constraints of the environment they are deployed in. Overall, the principle that underpins this work is that effective orchestration must be context-aware, domain-specific, and adaptive to real-time conditions, and towards this end, it combines system design, cross-layer monitoring, machine learning techniques, and software-defined architectures to deliver intelligent and practical solutions evaluated in realistic environments.

The thesis consists of seven main topics, each corresponding to a distinct contribution:

- **MAC-Layer-Based Diagnosis of WLAN Pathologies [1], [2]:** Chapters 2 and 3 provide the details of the first contribution that addresses the problem of performance degradation in residential and enterprise 802.11 WLAN deployments. It introduces an application-layer framework that is lightweight and exploits MAC-layer metrics exposed by the drivers of commercial access points to detect and classify common 802.11 pathologies, such as contention, low SNR, hidden terminal, and non-WiFi interference. The key novelty lies in the implementation of diagnostic mechanisms that can be both non-intrusive and accurate, based on signals from already available MAC statistics, such as packet delivery ratios and transmission attempts, without the need for external measurement hardware or firmware changes. In chapter 3, the employment of machine learning classifiers trained on experimentally collected data enables both passive and active detection modes that achieve high accuracy with minimal overhead.

- **Software Defined Networking in MANETs [3]:** Chapter 4 presents the integration of Software Defined Networking into MANETs to offer centralized intelligence and global optimization capabilities. Although SDN assumes stable connectivity to a centralized controller, an assumption that often fails in highly dynamic wireless environments, our implementation provides a fully-fledged scheme that incorporates an election mechanism for fault-tolerant controller establishment, dynamic topology discovery, and cross-layer decision-making based on MAC-layer link statistics. The resulting architecture bridges the gap

between distributed and centralized approaches and enables MANETs to benefit from SDN's centralized and cross-layer overview while still maintaining resilience to link volatility.

- **Reinforcement Learning for Adaptive SDN Configuration in MANETs [3]:** Building upon the work described in the previous chapter, chapter 5 provides a detailed analysis of the relevant system parameters of the SDN control plane and the trade-offs that accompany their configuration. Then, a reinforcement learning (RL)-based mechanism is introduced that autonomously tunes these system parameters in response to the underlying network conditions, trying to balance control-plane overhead with responsiveness. Evaluations show that this RL-enhanced system improves throughput while allowing quick adaptation of the SDN network to changes in physical layer connectivity.

- **Graph-Based Orchestration of Disaggregated Datacenter Resources [4]:** Our fourth contribution is presented in chapter 6, which addresses orchestration challenges in disaggregated cloud datacenters. In such architectures, compute and memory resources no longer reside in monolithic server trays but are aggregated in pools that are dynamically interconnected over programmable optical networks. Although this paradigm enables higher resource utilization and energy efficiency, it introduces new complexities in resource discovery, topology configuration, and workload scheduling. The thesis presents the design and implementation of a Software Defined Memory Controller that aims to overcome these challenges by modeling resources as a graph and allowing for a scalable and low-latency retrieval of configuration and connectivity information. The implementation is integrated with OpenStack and supports fine-grained orchestration of remote memory attachment, network reconfiguration, and workload placement, with evaluation results demonstrating significant performance gains for scaled deployments that are attributed to the efficient modeling and information storage.

- **Multi-Cluster Orchestration in 5G Experimental Testbeds [5]:** Chapter 7 introduces a practical orchestration framework for managing distributed experimental infrastructures using Kubernetes-based tools. As experimental 5G testbeds become increasingly federated and geographically dispersed, there is a need for flexible orchestration that maintains autonomy across the different administrative domains, while it still enables secure and high-performance inter-cluster connectivity. To this end, our proposed system combines SUSE Rancher for centralized management of multiple Kubernetes clusters and Submariner for establishing secure cross-cluster service exposure and interconnectivity. The framework is evaluated over a high-speed networking fabric (up to 25 Gbps) that links multiple testbed clusters, and demonstrates the ability to orchestrate containerized network functions (CNFs) and 5G workloads efficiently across federated environments. This contribution addresses practical challenges in multi-cluster orchestration, such as service federation, connectivity, and performance.

- **MLOps Framework for Heterogeneous Workflow Scheduling [6]:** Chapter 8 focuses on the orchestration of ML workflows across the computing continuum. Such workflows consist of multi-stage pipelines (e.g., data ingestion, training, and inference), each with its own requirements in regard to latency,

compute resources, and throughput. The thesis presents an MLOps framework that decomposes the workflows into functional containerized components and schedules them across heterogeneous resources. The implemented scheduler considers both performance requirements and infrastructure constraints and integrates with common container orchestration systems.

- **Energy-Efficient and Latency-Aware Scheduling of Inference Workloads [7]:** The final contribution is detailed in chapter 9. It addresses the challenge of scheduling ML inference workloads across the computing continuum by developing EELAS (Energy Efficient Latency-Aware Scheduling), a heuristic scheduling mechanism integrated into Kubernetes. EELAS continuously monitors available resources and dynamically places inference workload to minimize total energy usage while meeting latency constraints. Evaluation results or a real-world testbed exhibit substantial energy savings and improved execution time compared to baseline strategies.

Finally, Chapter 10 concludes this thesis by summarizing all the research contributions presented, as well as it describes several extensions as potential areas for future work.

Chapter 2

Enabling Wireless LAN Troubleshooting using MAC-layer Metrics

Contents

2.1 Introduction 7
2.2 Related Work 8
2.3 MAC-layer Statistics 9
2.4 IEEE 802.11 Related Pathologies 9
2.5 Detection Methodology 11
2.6 Experimentation with Proposed Metrics 12
2.7 Framework Evaluation 17
2.8 Conclusions and Future Work 18

Particular WLAN pathologies encountered in realistic scenarios are hard to detect due to the complex nature of the wireless medium. Previous efforts have employed sophisticated equipment, driver modifications, or application-layer techniques to diagnose such issues. The approach presented in this chapter focuses on identifying metrics that can characterize the root causes of individual pathologies, while being directly extractable from MAC-layer statistics available in modern wireless equipment. Through the development of a user-space framework on commercial hardware and its experimental validation, the effectiveness and applicability of this method are demonstrated.

2.1 Introduction

With home WLANs becoming increasingly popular and the plethora of wireless devices operating in the limited unlicensed spectrum, the performance degradation experienced by end-users is almost inevitable. Common home WLAN pathologies are related with low-quality channel conditions. However, even high quality links may suffer from anomalies that are inherent to the operation of the 802.11 standard, such as contention for medium access. In addition, the well-known 802.11 impairments of *"Hidden-Terminal"* [8] and *"Capture-effect"* [9], which are identified in dense topologies, frequently appear in closely spaced WLAN environments.

As administrators/users of home WLANs are not aware of such pathologies, performance issues are usually interpreted incorrectly and the blame is attributed to ISPs.

Troubleshooting WLAN performance is hard, due to the complex and dynamic nature of the wireless medium and requires collection of low-level information hardly interpreted even by experts.

Prior work in diagnosing wireless networks performance has considered a variety of approaches, ranging from in-depth studies [10, 11, 12] of specific pathologies through sophisticated equipment, to solutions [13] relying on vendor-specific drivers or modifications and application-layer frameworks [14] that are directly applicable to commercial WLAN devices. Considering the different categories of approaches, a tradeoff exists between the achievable detection accuracy and the applicability in common home WLAN setups. Towards bridging this gap, novel frameworks need to be developed that combine the advantages of both worlds.

This chapter presents the development of user-level detection mechanisms, which exploit low-level information that can be revealed by commercial Access Point (AP) devices. MAC-layer statistics are collected and updated as part of the Physical layer (PHY) rate adaptation mechanism. These statistics include, but are not limited to, the number of transmission attempts as well as the number of which were successful. The key novelty of our approach lies in the identification of metrics based on the aforementioned statistics that are able to characterize the root causes of WLAN pathologies. Through extensive experimentation, we concluded in the identification of unique trends that performance experiences, in terms of the proposed metrics, when 802.11 links are affected by different WLAN pathologies. Our detailed findings have been incorporated in a combined detection methodology that has been implemented on commercial APs from different vendors. The main outcome of our research is an application-layer framework that is automatically activated upon the detection of degraded performance to accurately determine the underlying pathology and report it to the end user. This study highlights the importance of having MAC-layer statistics accessible from the application-layer through a standardised way and encourage all manufacturers of 802.11 equipment to adopt this approach.

2.2 Related Work

A variety of research approaches have proposed mechanisms towards diagnosing common WLAN pathologies. Several works have focused on the detection of specific pathologies, such as distinguishing between frame losses resulting due to low signal or collisions in [10], or the identification of device types generating cross-technology interference in [12, 11, 15, 16]. Another class of approaches [17, 18] has proposed advanced anomaly detection frameworks that provide increased accuracy by combining measurements obtained from several nodes. In addition, [19, 20, 16] are based on the elaboration of multiple monitoring devices and require the application of synchronization protocols [20, 16], hence rendering them applicable only to centrally managed WLAN deployments. On the other hand, [21, 13] are based solely on observations derived from a single node, thus being applicable in independently owned home WLANs.

The various aforementioned approaches also differ on the specified implementation requirements. More specifically, in [12, 11] the use of sophisticated equipment is necessitated, while the approaches presented in [10, 21, 19, 13] require vendor-specific drivers or modifications. The main drawback of the aforementioned approaches is

that they are not hardware agnostic. In [14], the first user-level approach able to infer the MAC-layer effects of common home WLAN anomalies is proposed. This work takes a step further by developing a systematic approach able to detect the root causes of an extended list of pathologies, by taking advantage of the detailed information offered by MAC layer statistics, while still being accessible by the application layer.

2.3 MAC-layer Statistics

Commercial 802.11 devices that are developed by major vendors of wireless products, such as Atheros and Intel can be controlled through well-known Open-Source drivers (ath9k, iwlwifi, Mad-WiFi and ath10k [22]). Such drivers constantly collect detailed MAC-layer statistics that are updated as part of the PHY rate adaptation procedure, including among others, information related to the total number of attempted frame transmissions and retransmissions.

Based on this information we define and utilize two metrics. Firstly, the *Normalized Channel Accesses* : $NCA = CA/MCA$ where CA and MCA denote the attempted Channel Accesses and Model-Based Channel Accesses per second respectively, for a specific PHY rate and a specific frame length. MCA is calculated according to the 802.11 a/g performance model presented in [23]. We validated these values with experiments on various types of wireless chipsets under idle channel conditions at the fixed frame length of 1500 bytes. The NCA metric characterises the access (uninterrupted or not) to the wireless medium by a station willing to transmit data frames. Secondly, we define the *Frame Delivery Ratio* : $FDR = ST/CA$ where ST denotes the number of Successful Transmissions per second. The FDR metric is an indicator of the link quality which is responsible for the successful or not delivery of a frame.

In the following Sections, we present how these metrics can be exploited towards characterising the impact of commonly identified pathologies on WLAN performance.

2.4 IEEE 802.11 Related Pathologies

Perfomance of 802.11 stations first depends on the availability of channel access opportunities and second on the efficiency of frame delivery, whenever medium access is granted. The pathology identification mechanism is built upon this initial observation and categorizes pathologies into two classes. The first one considers pathologies occurring in cases where the transmitter identifies the medium as busy and thus defers from transmitting (**Medium Contention**). In the second category, we group pathologies occuring in cases that the medium is detected as idle, thus enabling the transceiver to proceed with frame transmissions that fail to be delivered at the receiver (**Frame Loss**). The taxonomy of the considered pathologies is presented in Fig. 2.1.

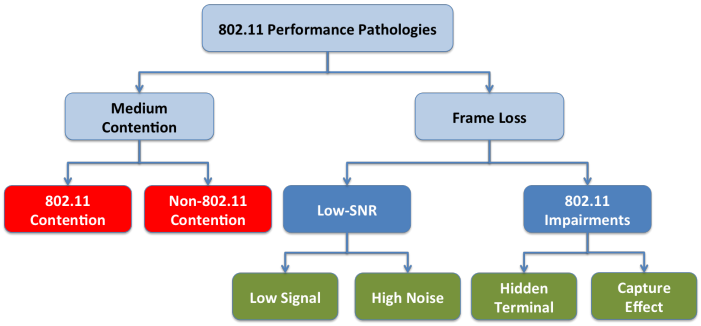


FIGURE 2.1: Taxonomy of IEEE 802.11 Pathologies

2.4.1 Medium Contention

Contention-based pathologies frequently occur in dense WLAN deployments, where multiple 802.11 devices concurrently attempt to access the medium. However, as the unlicensed spectrum is also exploited by other wireless protocols (e.g. Bluetooth, Zigbee) and a large range of RF devices (e.g. cordless phones, security cameras), the medium is further congested due to non-802.11 emmissions. The resulting decrease in available channel access opportunities is directly dependent both on the channel airtime captured by 802.11 transmissions, as well as the transmission Duty Cycle (DC) of non-802.11 RF devices.

The crucial impact of medium contention is clearly highlighted in cases of contending stations that utilize diverse PHY rate configurations, which leads to performance anomaly [24]. The high bitrate stations observe a higher throughput degradation in comparison with the lower rate nodes. This degradation is a result of the low number of CA attempts due to the high channel airtime utilization by the low bitrate stations. Consequently, the NCA metric is expected to decrease across increasing PHY rate configurations of the concerned station. However, regarding the FDR metric, higher bitrates should result in higher number of collisions, due to simultaneously expiring back-off timers, and thus to a decrease in FDR, but not in that extent of considering it as a significant trend.

In case of non-802.11 contention, devices with fixed transmission DC, such as microwave ovens, can be interpreted as low bitrate stations which do not comply with the 802.11 standard and hence do not perform a *"Backoff"* procedure. As a result a decrease in NCA metric is expected across increasing PHY rates, as it happens in 802.11 contention. Another consequence of the absence of backoff mechanism in non-802.11 devices is that collisions can occur in the middle of a frame transmission and so higher PHY rates will result in lower probability of collisions. Taking that into consideration, we expect an increasing trend in the FDR metric.

2.4.2 Frame Loss

In this category, we group pathologies generated in scenarios that the 802.11 *"Channel Sensing"* mechanism constantly identifies the medium to be idle and grants uninterrupted medium access. However, conditions experienced at the receiver side may lead in reduced probability of successful Frame Delivery and subsequent doubling of the Contention Window (CW) parameter. As the reduced MAC-layer Frame Delivery Ratio (FDR) is the root cause of this phenomenon, we identify it as the key metric for characterizing the impact of Frame Loss related pathologies.

Fundamental causes of receiver side underperformance are usually related with the *low-SNR conditions* experienced as a consequence of the low Received Signal strength, resulting from channel fading and shadowing or due to high-Power non-802.11 emissions that result in Noise level increase. Considering that complex modulation schemes require higher link SNR to ensure reliable communication, in comparison with basic schemes, we expect the FDR performance to significantly decrease across increasing PHY rates, under low-SNR conditions. Furthermore, the decrease in FDR would also lead in an decrease in the NCA metric, as the doubling of the CW results in fewer CAs.

In addition, significant frame delivery inefficiencies may also be attributed to *802.11 impairments*, phenomena appearing in cases that concurrent channel access and subsequent frame collisions cannot be avoided through the 802.11 *Channel Sensing* mechanism. More specifically, the *"Hidden-Terminal"* anomaly occurs in cases that the receiver node lies within the transmission range of two active 802.11 nodes that are mutually hidden and cannot sense each other resulting in frame collisions. In cases that no remarkable difference is observed in the received signal strength of colliding frames at the intermediate node, the *"Hidden-Terminal"* phenomenon appears symmetrically for both flows. However, the most frequently observed case is the *"Capture-effect"* phenomenon, in which case a considerable difference in RSSI values is observed, resulting in a higher probability of successful decoding for the high-power frames. As a result, the link *"capturing"* the medium experiences lower collision probability accessing the medium more frequently and resulting in higher performance penalty for the affected links.

Longer duration transmissions experiencing higher probability of collision, so we expect to see an FDR increase across increasing PHY rate values of the affected link. However, hidden nodes suggest longer distances from the AP and consequently an underlying low-SNR pathology, so we also expect an FDR decrease in high PHY bitrates. In overall, we should identify a highly varying FDR metric accross PHY rates and additionally more notable variations under *"Capture-effect"* scenarios where the impact is more severe. As regards the NCA metric, although the underlying low-SNR conditions should impose a decreasing trend, the impact of FDR variation, which as mentioned before is higher under *"Capture-effect"* scenarios, would enforce NCA to not display a clear trend.

2.5 Detection Methodology

Having defined the key metrics of NCA and FDR, we next focus on developing a detection methodology able to identify unique trends on the way each individual pathology affects performance of both metrics. Before that, we need to decide upon the existence or not of a pathology. This is accomplished by a simple throughput test of fixed length frames at the maximum PHY rate, the result of which is compared to the analytical value calculated by the aforementioned model in [23]. In case that achieved throughput is lower than the 80% of the theoretical one, we initiate our proposed framework presented below. By taking advantage of the relation between the PHY rate of the affected link and the proposed metrics, we design an active probing mechanism that probes the WLAN channel with multiple packet trains, where each train consists of several packets that are transmitted at varying PHY rates. We call the proposed test as *Varying Bitrate Probing* and each train is

transmitted in each one of the 802.11a/g compatible PHY rates, selected from the vector $R = (6, 9, 12, 18, 24, 36, 48, 54)$ Mbps. Each train provides a unique sample - we need multiple samples to make any statistical inference. In parallel with the probing procedure, the NCA and FDR metrics are calculated per each configured PHY rate.

Next, we apply the non-parametric *Theil-Sen Slope* estimator on the collected samples to identify trends in the relation between the two metrics and the PHY rate of the affected link. The output of the *Theil-Sen Slope* estimator consists of the slope estimation with 95% confidence interval, plus the p-value, where both aid in determining the existence of a trend and its characterization as increasing or decreasing. P-values are interpreted as follows: $p < 0.01$ indicates very high significance and $p < 0.05$ is considered significant and the null hypothesis (of the slope being equal to zero) is rejected in both cases. P values greater than 0.05 indicate failure to reject the null hypothesis and thus no trend is detected.

In such occasions, we have to distinguish between two further cases, where in the first case, data points present highly varying values and partially present both significantly increasing and decreasing trends, while in the second the considered input data is roughly constant and result in approximately zero estimated slope. Although in the first case, no specific trend can be reliably detected, several scenarios might present high start-to-end variation, a trend that we also need to identify. To this aim, we enhance our test, by employing the *Pairwise Difference Test* metric $S_{PDT} = \frac{FDR^8 - FDR^1}{\sum_{k=2}^{8} |FDR^k - FDR^{k-1}|}$, where $k \in \{1, 2, ..., 8\}$ denotes the configured PHY rate. It is obvious that $-1 \leq S_{PDT} \leq 1$. If there is a strong trend, either increasing or decreasing, S_{PDT} approaches 1 or -1. Identification of the second case is based on the evaluation of statistical dispersion through the measure of standard deviation. We consider specific standard deviation thresholds, as derived from our experimentation and described in the following section.

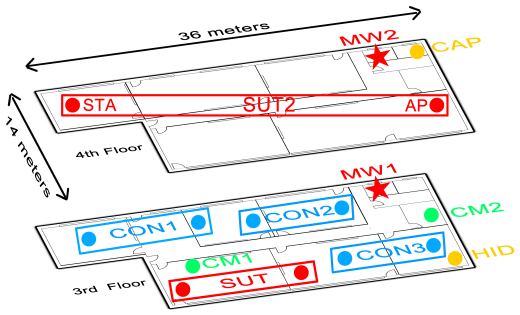
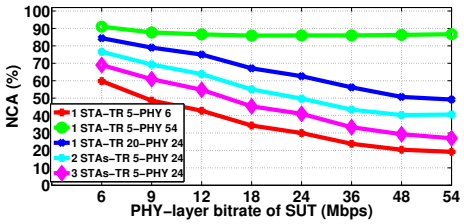


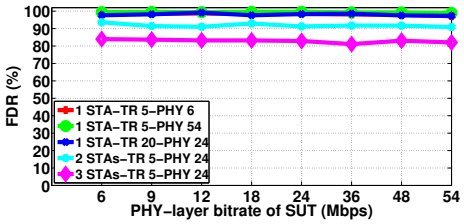
FIGURE 2.2: Testbed Topology

2.6 Experimentation with Proposed Metrics

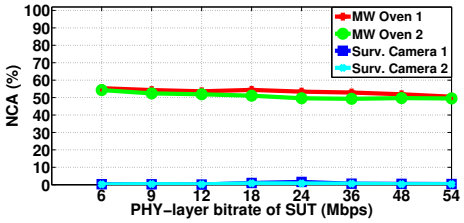
The experimental setup that is used as the basis of our experimentation consists of a single communicating pair of nodes that we refer to as System under Test (SUT). Both nodes feature the Intel 5300 chipset, implement the 802.11a/g protocol and operate in infrastructure mode, through the *iwlwifi* driver. In the following experiments, we reproduce each considered pathology and investigate how the performance of the SUT link is affected in terms of the NCA and FDR metrics, while it performs the *Varying Bitrate Probing* test. The devices participating in the following



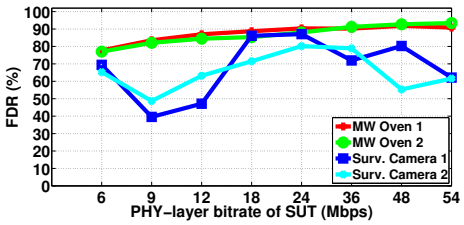
(A) 802.11 Contention NCA



(B) 802.11 Contention FDR



(A) non-802.11 Contention NCA



(B) non-802.11 Contention FDR

FIGURE 2.4: NCA and FDR Performance of Medium Contention related Pathologies

experiments are closely located within a double floor indoor office environment at the University of Thessaly premises, as depicted in Fig. 2.2. A representative subset of the various executed experiments that replicate each individual pathology is detailed in the following Sections.

2.6.1 Contention with 802.11 terminals

Through this first experiment, we aim at investigating the impact of medium contention with 802.11 compatible devices, and for this purpose we establish 3 contending links in close proximity and on the same channel with the SUT link. More specifically, we use Ch. 44 of the 5 GHz band that is totally free of transmissions in the testbed premises.

In the first 2 Scenarios, we activate only the CON1 link to transmit 5 Mbps of traffic load, at the PHY rates of 6 Mbps and 54 Mbps accordingly. Fig. 2.3a depicts the NCA performance and shows a significant decreasing trend across increasing PHY rates in Sc.1. due to the *"802.11 performance anomaly"*, while in Sc. 2 only minimal variation is detected across increasing PHY rates, as a result of the high PHY rate. In Sc. 3, we still activate only the CON1 link to transmit at the PHY rate of 24 Mbps with 20 Mbps of traffic load. We observe that in Sc. 3, the NCA values per PHY rate have decreased in comparison with Sc. 2, while a significant decreasing trend is clearly identified. Finally, in Sc. 4, we simultaneously activate links CON1 and CON2 to transmit 5 Mbps of traffic load, at the PHY rate of 24 Mbps, while in Sc. 5 we replicate the configurations of Sc. 4, but simultaneously activate the 3 links CON1, CON2 and CON3. In both cases, significant decreasing trends are identified by the Theil-Sen estimator. Summarising the above scenarios, a significant decreasing trend is detected with a p-value of 0.01 except for the Sc. 2 where the p-value of 0.4 is derived and thus no trend is detected. Fig. 2.3b plots the resulting FDR performance across all the considered Scenarios and presents only minimal variation across different PHY rates (standard deviation of 0.95). We notice that the increasing number of contending stations results in decreased FDR, as also observed in [25], a fact related with the increased probability of collisions when the back-off timers of multiple terminals simultaneously expire.

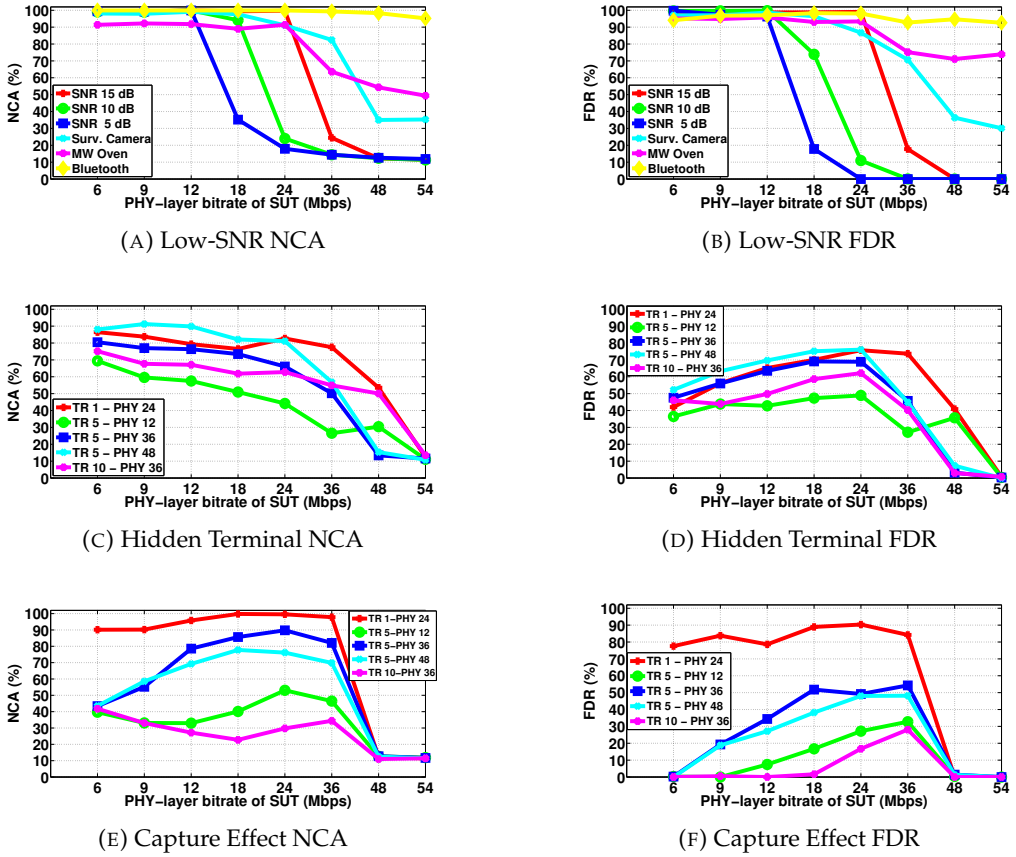
2.6.2 Contention with non-802.11 devices

In this second experiment, we aim at characterising the impact of different types of non-802.11 devices. More specifically, we consider a Microwave Oven (MW) that typically emits high RF energy in 2.44-2.47 GHz frequencies with DC of 0.5 and a Surveillance Camera that constantly (DC = 1) transmits with 10 dBm power, occupying 18 MHz of bandwidth on various frequencies of the 2.4 GHz band. The two devices are located at positions MW1 and CM1 of the 3rd floor accordingly. We set the SUT link to operate on the commonly configured Ch.6 (2437 MHz) and the camera on 2432 MHz.

In Fig. 2.4a, we clearly observe that the continuously emitting Surv. Camera results in NCA values that are close to zero across PHY rates, as the SUT link constantly detects the medium to be busy. On the other hand, the MW that is activated with the DC of 0.5, provides a fixed amount of time available for medium utilisation per period. This phenomenon affects performance in terms of the NCA metric, in a way similar to the *"802.11 performance anomaly"*, thus leading the NCA values to decrease across increasing PHY rates. The resulting FDR performance is depicted in Fig. 2.4b, where in the case of the MW, an increasing trend is observed with the p-value of 0.01. Considering the FDR evaluation of the Surv. Camera, no specific trend is identified, as the FDR highly fluctuates due to the extremely low number of attempted transmissions.

2.6.3 Low SNR

In this experiment, we jointly investigate the impact of low-SNR conditions resulting either in *Low Signal* or *High Noise* scenarios. Considering the *Low Signal* case, we generate varying low-SNR topologies, by establishing a remote 802.11 link (SUT2)



(A) Low-SNR NCA

(B) Low-SNR FDR

(C) Hidden Terminal NCA

(D) Hidden Terminal FDR

(E) Capture Effect NCA

(F) Capture Effect FDR

FIGURE 2.5: NCA and FDR performance under frame loss related pathologies: Low-SNR, Hidden Terminal, and Capture Effect.

on the 4th floor and properly tuning the transmitter's Power, to result in links of 15 dB, 10 dB and 5 dB SNR. In Figures 2.5a and 2.5b, we observe that performance regarding NCA and FDR is unaffected as long as the SNR provides for robust communication at the given PHY rate. However, in cases where the SNR requirement of the configured bitrate exceeds the SUT's link SNR, a remarkable drop in NCA and FDR is noticed resulting in a decreasing trend (p-value 0.01) for both metrics. Towards experimenting under High Noise conditions, we use the SUT link of the 3rd floor and activate the MW at position MW2 of the 4th floor, along with the Surv. Camera at position CM2 of the 3rd floor. In both scenarios, the high power emissions of the remotely located interfering devices are not detected to exceed the high Energy Detection threshold at the location of the SUT link. However, due to their high DC, the *Noise level* is constantly increasing, hence generating low-SNR conditions and approximating the performance obtained in *Low Signal* scenarios, in terms of both metrics.

2.6.4 Hidden Terminal

Towards experimenting with *"Hidden-Terminal"* scenarios, we establish a hidden to the SUT2 link, by activating the transmitter at position HID of the third floor and resulting in equally received signal strength at both link receivers. In the first scenario we measure the impact of 1 Mbps traffic load, transmitted at the PHY rate of 24 Mbps. As shown in Fig. 2.5c, the NCA metric presents a decreasing trend,

across increasing PHY rates, while no specific trend is identified for the FDR metric, as depicted in 2.5d. More specifically, a small increase across the first PHY rates is followed by a sharp decrease due to the underlying low-SNR pathology. In the next three scenarios, we fix the traffic load at 5 Mbps and vary the PHY rate of the hidden link, between 12, 24 and 48 Mbps respectively. Across all the tested scenarios, identical trends as in Sc. 1 are detected. In Sc. 5, we fix the PHY rate of 36 Mbps and transmit at the traffic load of 10 Mbps, noticing similar performance as in previous scenarios. In all scenarios a decreasing trend is detected regarding the NCA metric with p-value of 0.01, while the null hypothesis regarding FDR is rejected (p-value $>$ 0.05).

2.6.5 Capture Effect

In this last set of experiments, we examine the performance fluctuations under various scenarios experiencing the *"Capture-effect"* phenomenon. For this purpose, we use the node located at position CAP of the 4th floor, as the interferer of the SUT2 link. In Sc.1, we start by injecting the light traffic load of 1 Mbps in the wireless medium, while configuring the interfering link at the PHY rate of 24 Mbps. As presented in Fig. 2.5e and Fig. 2.5f, similar trends are observed as in the considered *"Hidden-Terminal"* scenarios. In Scenarios 2, 3 and 4, we increase the traffic load of the interfering link to 5 Mbps and vary the PHY rate among 12 Mbps, 36 Mbps and 48 Mbps, while in Sc. 5 the traffic load is further increased to 10 Mbps and the PHY rate is fixed at 36 Mbps. Across all the considered scenarios, the NCA metric presents no significant trend, as the calculated p-values lie above 0.05. As regards to FDR, no trend is detected in all Scenarios (p-values above 0.23), although the high performance penalty in comparison to *"Hidden-Terminal"* is depicted with close to zero FDR values. Regarding the performance obtained in Sc. 1, we remark that as both 802.11 impairments pose similar impact on both metrics under low traffic load conditions, discrimination between the two phenomena will be challenging under such cases.

2.6.6 Framework Enhancement and Results Summary

In many cases of frame loss pathologies, Theil-Sen estimator falsely concludes that the existing pathology is the hidden terminal one. For that reason, the introduction of the aforementioned PDT metric enhances our test with further refinement of the identification of trends. More specifically, in cases where the Theil-Sen estimator detects decreasing trend in NCA attempts and no trend in FDR metric, we apply the PDT metric in FDR statistics. Through extensive experimentation, we concluded that $S_{PDT} <$ -0.8 denotes low-SNR pathology, -0.8 $\leq S_{PDT} \leq$ -0.32 denotes hidden terminal pathology and $S_{PDT} >$ -0.32 denotes capture effect pathology. The outcome of our study is presented in Table 2.1, which lists the specific trends that can be detected through the proposed methodology. Not every combination of metrics' trends is mapped to a specific pathology and that cases may correspond to the existence of multiple simultaneous pathologies, which we do not consider in our current work. The derived findings have been incorporated in an application-layer framework that is automatically activated upon the detection of degraded performance to uniquely determine the underlying pathology.

		Frame Delivery Rate (FDR)			
		Constant	No Trend	Increasing	Decreasing
Normalized Channel Accesses (NCA)	Decreasing	802.11 Contention	Hidden Terminal	Non-802.11 Contention DC < 1	Low SNR
	No Trend		Capture Effect		
	Constant		Non-802.11 Contention DC = 1		

TABLE 2.1: Identified Trends per considered Pathology

2.7 Framework Evaluation

In this section, we extensively evaluate the detection performance of the developed framework, under two specifically designed sets of experiments. In each set, we replicate a specific anomaly under various scenarios and by measuring the number of true positives we quantify the perceived detection accuracy. In the first set, we generate scenarios of contention with 802.11 devices, while in the second we experiment with low-SNR scenarios and 802.11 impairments. For the sake of completeness we also examine cases, where our throughput test does not consider as pathologies.

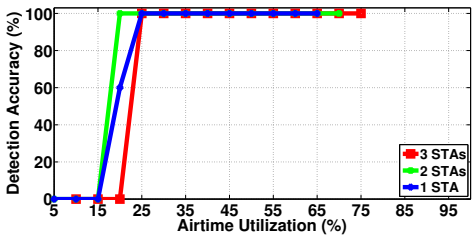
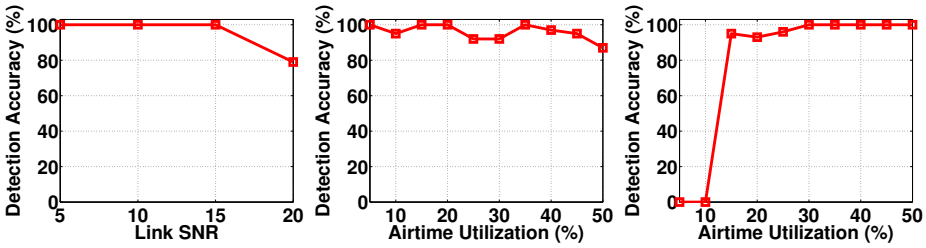


FIGURE 2.6: Detection Performance across different 802.11 contention scenarios.

2.7.1 Contention with 802.11 terminals

We start by configuring 3 different topologies, consisting of 1, 2 and 3 contending stations that coexist within the 3rd floor of the testbed. We replicate 36 different scenarios in each different topology (108 in total), by varying both the configured PHY rate and traffic load parameters to generate diverse medium utilisation conditions. Through an extra wireless node, we monitor the percentage of Airtime that is captured by the contending links in each topology, towards highlighting the impact of Airtime Utilization on the resulting detection accuracy. In Fig. 2.6, we clearly observe that the detection performance improves across increasing medium utilisation conditions as the impact of contention is becoming more evident. More specifically, as medium utilisation increases above 25% the mechanism successfully detects the 802.11 contention pathology, across all the corresponding scenarios in the 3 different topologies. Low medium utilisation conditions (below 15%), however, are always detected as no pathology by the initial throughput test and thus are not considered as detection failures.



(A) Low-SNR (B) Hidden Terminal (C) Capture Effect

FIGURE 2.7: Detection performance across Frame Loss related pathologies.

2.7.2 Frame Loss

In this set of experiments we evaluate the detection performance under low-SNR scenarios, by placing the transmitter of the SUT link at the 3rd floor and the receiver across 20 different locations at the 4th floor. We also vary the transmission Power of the SUT transmitter to further vary the SNR levels in each link and result in 80 different topologies. For each different topology, the SUT link executes our detection mechanism, in order to investigate whether the low-SNR conditions are detected. The resulting scenarios are grouped in 5 different SNR classes. We observe in Fig. 2.7a that the detection accuracy is 100% for all SNR classes, except for the 25 dB case, which poses no significant impact and is not detected as pathology from the initial throughput test.

Towards replicating the Hidden-Terminal and Capture Effect phenomena, we activate an interfering link at a fixed position in the 4th floor which is hidden to the transmitter of the SUT. By observing the RSSI values of the transmitted frames, we notice that 4 of the topologies lead to nearly equal (approximately 0-3 dB of difference) values between SUT's transmitter and interfering link's transmitter and consequently are vulnerable to the hidden terminal pathology. Furthermore, 9 links present a notable (>15 dB) difference in RSSI values and hence are vulnerable to the capture effect phenomenon. We evaluate our algorithm in the corresponding topologies that potentially suffer from 802.11 impairments, while inducing traffic of varying load and PHY bitrate at the interfering link and consider 36 different scenarios for each topology. In Fig. 2.7b, we observe that the Hidden Terminal pathology is successfully detected across the various tested scenarios that are presented in order of airtime utilised by the Interfering link. In the case of the capture effect pathology, we notice in Fig. 2.7c that the obtained accuracy presents low performance for low airtime utilisation, due to the pathology causing similar impact upon the suffering nodes as the *"Hidden-Terminal"* one does.

2.8 Conclusions and Future Work

The proposed detection framework of WLAN pathologies causing performance degradation showed encouraging results by accurately detecting all the considered pathologies. Our approach of utilizing the MAC-layer statistics offered from some wireless devices' vendors pointed out the importance of making these accessible, as they are already implemented, to user-level. As we demonstrated this will be

of great advantage to WLANs administrators in their effort of troubleshooting low performance. Future extensions of this diagnostic framework could involve deployment across volunteer access points to assess its robustness in diverse home network environments. Additionally, enhancements are needed to detect multiple simultaneous pathologies, which remains a limitation of the current implementation.

Chapter 3

On the Employment of Machine Learning Techniques for Troubleshooting WiFi Networks

Contents

3.1 Introduction **21**
3.2 Related Work **23**
3.3 Methodology **23**
3.4 Evaluation of classification algorithms **28**
3.5 Conclusion **31**

The rapidly increasing popularity of 802.11 WLANs along with the co-existence of multiple heterogeneous devices in the unlicensed frequency bands have created unprecedented levels of congestion, especially in densely populated urban areas. Under such complex setups, WLAN under-performance issues experienced by end-users are hard to interpret even by experts. This chapter presents the design and evaluation of an intelligent and easily deployable mechanism that leverages MAC-layer statistics and applies machine learning techniques to diagnose five of the most prevalent WiFi pathologies, including medium contention, low signal-to-noise ratio (SNR), and non-802.11 interference. The proposed framework incorporates data collection and analysis components, which feed into four classification algorithms. These models are carefully fine-tuned to optimize their hyper-parameters with respect to detection precision and overall accuracy. The resulting system offers two complementary diagnostic modes: a passive detection mechanism with minimal overhead, and a more accurate active probing approach. Through extensive testbed experimentation, the detection performance of the proposed methods is assessed. Results indicate that the K-Nearest Neighbors classifier achieves near-perfect accuracy and precision—approaching 100% for active probing and approximately 95% for passive detection—across all five considered pathologies.

3.1 Introduction

The IEEE 802.11 standard, commonly known as WiFi, has grown tremendously and become ubiquitous in both home and enterprise environments. Private and public WiFi networks now carry a substantial portion of IP traffic—45% as of recent estimates—with projections indicating an increase to 49% by 2020 [26].

The ongoing adoption of Internet of Things (IoT) technologies—such as smart TVs, wearables (e.g., smartwatches, activity trackers), security cameras, energy monitors, and smart lighting—has further expanded the scale and complexity of home WLANs. These devices, while often based on different protocols (WiFi, ZigBee, Bluetooth), must coexist within a shared and limited unlicensed spectrum, particularly in dense urban environments. This coexistence can result in degraded WiFi performance due to medium contention or interference from non-WiFi devices, including common appliances like microwave ovens. In addition, improper placement or configuration of Access Points (APs) by inexperienced users may lead to suboptimal channel conditions. The challenge is further exacerbated by inherent 802.11 protocol limitations such as the Hidden Terminal and Capture Effect phenomena. As a result, end-users frequently experience performance degradation, which is difficult to diagnose without technical expertise.

Troubleshooting WiFi performance issues is inherently complex due to the dynamic and opaque nature of the wireless medium. Accurate diagnosis typically requires access to detailed protocol-level information that is challenging to collect and interpret—especially for non-experts. In enterprise networks, dedicated network administrators can draw on specialized equipment, knowledge of the topology, and protocol expertise to diagnose problems effectively. In contrast, home users lack these resources and must address similar challenges with limited tools and expertise. These constraints underscore the growing need for automated, accessible mechanisms capable of identifying the causes of performance degradation in modern WLAN environments.

This chapter builds upon previous work [1], which was presented in the previous chapter and demonstrated the feasibility of equipping low-cost commercial APs with user-level mechanisms to detect WiFi performance impairments and identify the root pathologies. Here, we extend that framework by employing a data-driven approach [27] using MAC-layer information collected by wireless drivers as part of the rate adaptation process. Specifically, we focus on transmission attempts and the success rate of transmissions—metrics that are readily available at user level and informative regarding the health of a WiFi link. These metrics form the foundation for diagnosing five key WiFi pathologies.

The chapter evaluates the use of four widely recognized classification algorithms—*Decision Trees*, *Random Forests*, *Support Vector Machines*, and *K-Nearest Neighbors*—for identifying pathologies based on collected data. The algorithms are trained and fine-tuned using a comprehensive dataset obtained from controlled testbed experiments in which traffic is injected to simulate a wide range of scenarios. The resulting classifiers are evaluated on both accuracy and precision. We introduce two diagnostic modes: one that supports passive monitoring and another that relies on active probing. This dual-mode design addresses the limitations of earlier active-only methods and enables deployment scenarios with minimal overhead.

Our objective is to deliver a lightweight, intelligent framework suitable for deployment on commodity APs, capable of accurately diagnosing common WLAN pathologies. This work contributes a practical, user-level solution that supports both passive and active modes and covers a broad set of performance issues.

The remainder of this chapter is organized as follows. Section 3.2 reviews related work. Section 3.3 presents our data collection methodology. Section 3.4 evaluates the performance of the classification models. Finally, Section 3.5 concludes the chapter

and discusses potential future directions.

3.2 Related Work

There are a few similar approaches that have been followed in the literature for detecting causes of poor performance in WiFi networks. As an example, the work by Kanuparthy et al.[14] takes advantage of user-level information for distinguishing between the different 802.11 pathologies, by employing active probing and estimating simple metrics such as transmission rate. WiSlow, [28] also exploits MAC-layer information gathered after injecting traffic to the network for discriminating between 802.11 and non-802.11 interference with a high accuracy, when interfering devices are close to the suffering node. However, our work differentiates by providing a mode of passive pathology detection that does not incur any additional overhead and moreover does not impose any significant accuracy penalty.

Machine learning techniques have been heavily employed in recent years, when abundance of data exists. WiFi is not an exception, with several frameworks being developed for either estimating key performance indicators of WLANs or classifying sources of underperformance. In [29], authors apply the classification algorithms, considered also in our work, for characterizing and estimating WiFi latency. Artificial Neural Networks are used in [30] for estimating packet delivery rate, while a hidden Markov model estimates the probability of interference between 802.11 nodes from traces collected from multiple sniffers in [31].

In addition, several works have put an effort on investigating the detection and classification of the various causes of poor performance in WiFi networks with the application of machine learning algorithms. However, most of them have focused basically on the identification of interference, especially cross-technology, using statistics gathered from commodity hardware or traces from monitoring devices. A notable example of such a work is [15], where authors feed results of the spectral scan function performed by Atheros wireless cards to Decision Tree and SVM classifiers, in order to distinguish between non-WiFi devices sharing the unlicensed spectrum. The work in [32], based on information provided by commercial cards (sequences of receiver errors), aims at detecting sources of non-WiFi interference by the employment of Artificial Neural Networks and hidden Markov chains. In [33], authors employ Machine Learning techniques for accurate interference estimation, while cross-technology interference detection has also been considered in 802.15.4 (Zigbee) networks[34],[35].

In contrast to the aforementioned body of work, our detection methodology takes a step further and considers all the potential pathologies. It covers 802.11 Contention, non-802.11 interference, low Signal-to-Noise ratio and specific to 802.11 anomalies, such as the Hidden Terminal and Capture Effect phenomena [1], thus providing more insight on the underlying causes of underperformance.

3.3 Methodology

We deploy our detection framework as a script running on the AP that implements two different mechanisms for sampling our defined metrics, an active and a passive

one, both of them presented in the rest of the section. The samples are then given as input to a classification algorithm, which we have selected according to the analysis provided in Section 3.4.

3.3.1 Sampled metrics

As in [1], we define the key metrics that are able to characterize WiFi pathologies based on the two key factors mentioned in Section 2.4 that impact 802.11 performance: a) transmission attempts and b) efficiency of the transmissions. We represent transmission attempts with the *Normalized Channel Accesses (NCA)* metric, defined as:

$$NCA = \frac{CA}{MCA} \tag{3.1}$$

where CA denotes the number of channel accesses per second, while MCA represents the maximum number of channel accesses per second, as are analytically calculated from a performance model of 802.11. On the other hand, we represent the transmission efficiency with the *Frame Delivery Ratio (FDR)* metric, defined as:

$$FDR = \frac{ST}{CA} \tag{3.2}$$

where ST denotes the number of successful transmissions per second.

When the channel is sensed as idle and there is correct reception of frames on the receiving end, performance is ideal, thus NCA and FDR values approximate to 1. However, when channel access opportunities are limited, NCA values drop significantly. Restricted access to the wireless medium can occur in two cases. Either due to the channel being sensed as busy, so transmissions are infrequent but still successful, meaning that FDR remains unaffected, or due to frame losses/collisions that subsequently decrease FDR values.

The aforementioned metrics are sampled by logging their values, as these are exported by the driver of the wireless card, ath9k, as part of Minstrel rate control algorithm, to the debugging file system. During experimentation described in Subsection 3.3.4, the logs of the metrics' values were being transferred to an external server, where we performed the training and testing phases of the considered classification algorithms.

3.3.2 Active probing

The sampling of the previously defined metrics follows the same methodology as in [1], as we employed the *Varying Bitrate Probing* mechanism. By taking advantage of the relation between the PHY rate of the affected link and the proposed metrics, this active probing mechanism probes the wireless channel with multiple packet trains, where each train consists of several packets of a fixed length that are transmitted with all of the available single stream modulation and coding schemes (MCS) of 802.11 n protocol $\{MCS0, MCS1, ..., MCS7\}$. We have implemented *Varying Bitrate*

Probing on commercial AP devices, where we installed OpenWrt, a lightweight operating system for embedded devices and created a script for generating saturated traffic transmitted with a specific modulation to a device acting as a station (STA).

3.3.3 Passive monitoring

The rate control algorithm used by the ath9k driver, Minstrel, is based on statistics and more specifically, on the probability of successful transmission for determining the rate that provides the best throughput. Evidently, in order to acquire these statistics, it needs to sample all of the available MCS. This is done by randomly selecting the 10% of the transmissions as sampling transmissions for evaluating a random MCS. We, in turn, take advantage of this fact, to sample *NCA* and *FDR* values without the need for injecting probing traffic on the network. We sample every 60s, as long as there are transmissions ongoing. This, rather large, sampling window allows time for more transmissions and subsequently more updated values. It is worth noting that if there is no or little user-initiated traffic, then there are not enough samples. However, we argue that in that case there is no need for troubleshooting, as there is no problem noticed.

3.3.4 Experimentation in controlled environment

In order to reproduce all of the considered pathologies and generate numerous data samples for feeding classification algorithms, we performed several experimental sessions for each pathology in our testbed. Our setup consisted of an OpenWrt enabled wireless router, TP-Link TL-WR2543ND, featuring the AR9380 chipset, operating on both the 2.4 GHz and 5 GHz bands, supported by the ath9k driver, where *Varying Bitrate Probing* was implemented. Traffic was sent to a smartphone device acting as a STA that enabled us to replicate a great variety of scenarios and topologies. In order to replicate, the scenarios of contending and hidden nodes, we employed several wireless nodes available in our testbed. Our evaluation link, between the AP and the smartphone is configured on channel 36 of the 5 GHz band which is free from external interference in our testbed.

802.11 Contention

We first considered the case, where multiple 802.11 devices operate in the same frequency within each other's range. We established three different wireless links operating on the same channel and randomly activated each one, in order to generate multiple contention scenarios. We also alternated the traffic rate (1, 2, 10, 24 and 50 Mbps) and the MCS of the links for covering as much as possible of realistic WiFi activity. In total, we collected 928 samples. As also stated in [1], we observed that due to an 802.11 performance anomaly, higher MCS suffer more when there is contention from "slower" devices, thus heavily impacting the corresponding *NCA* values. On the other hand, *FDR* values remain rather stable and high as there are no frame losses.

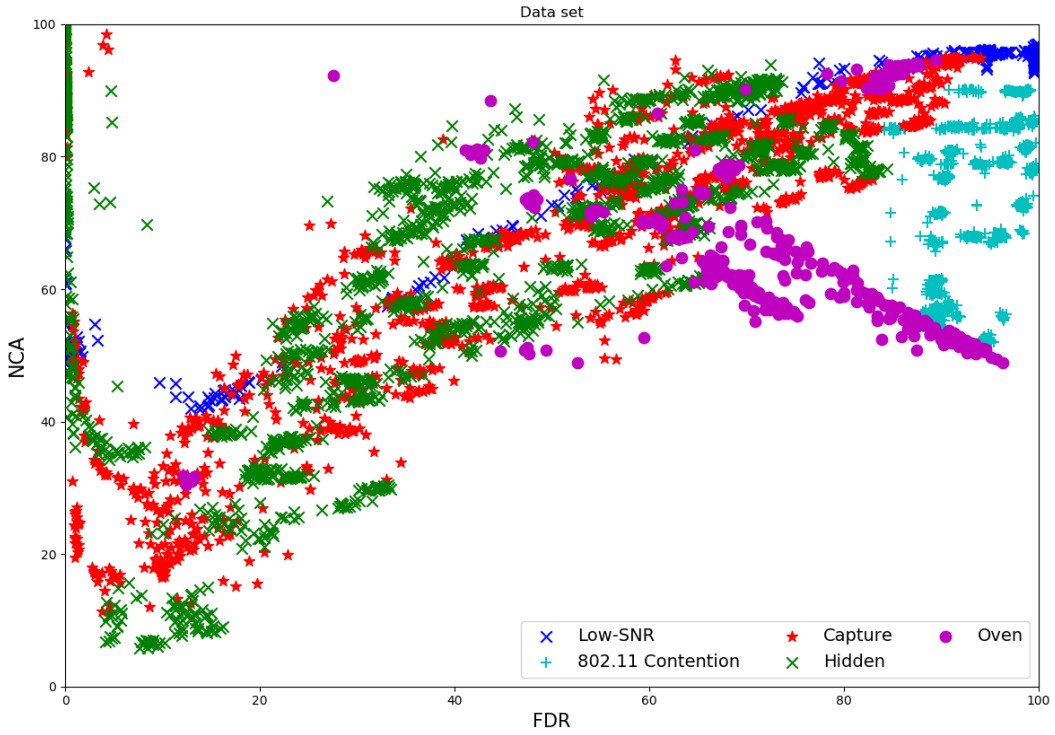


FIGURE 3.1: Data set.

Non-802.11 Contention

A microwave oven was selected as the non-802.11 device with which we generated non-802.11 contention samples. This is a high RF energy emitting device, found in almost every home and can have a serious effect on performance as it operates in 2.44-2.47 GHz frequencies. For this reason, we configured our evaluation link on channel 7 only for this scenario. The operation of the microwave oven can be considered as similar to a "slow" WiFi STA that emits energy with a duty cycle of 0.5. However, as it does not follow the exponential backoff policy, it starts its emissions without sensing the medium, thus causing unrecoverable errors on simultaneous transmissions of WiFi devices. We performed experiments in various locations away from the MW oven resulting in the gathering of 390 samples. As it follows from the above description, *NCA* exhibited similar performance across MCS, while *FDR* in higher modulation schemes dropped due to the greater number of collisions.

Low SNR

In this set of data sampling experiments, we replicated various low SNR scenarios for both cases of low signal power received on the evaluation and high RF noise existence close to the receiving smartphone. For the first case, we positioned the smartphone in 20 locations ranging from 1 to 40 meters, resulting in various SNR values. In addition, we alternated the transmission power of the AP device between 0, 5, 10 and 15 dBm. As regards the case of high noise, we employed a wireless surveillance camera at 10 different locations, outside the AP's range. In total, we collected 1087 samples. By inspecting the data, we observed that higher MCS, which

are less robust, dropped their *FDR* values and as a result from frame losses and the subsequent increase of the CW, a similar behavior was observed for *NCA*.

Hidden Terminal

A more complex pathology to replicate was symmetric Hidden Terminal. In this case, frames coming from both the transmitters of the evaluation and the interfering link respectively, should have similar RSSI value, in order to result to frame loss for both after a collision. We gathered data from 10 locations and in a similar manner to contention scenarios, we varied the traffic rate and the modulation scheme of the interfering link, accumulating a total of 1408 samples. It is worth noting that in order to achieve having the transmitters outside each other's range, the distance between the evaluation link was long, thus we anticipated the coexistence of the *Low SNR* pathology. This explains the observation we made from analyzing the data that *FDR* improves as the modulation scheme gets higher, and so does the *NCA*, because smaller duration of frame transmissions leads to avoidance of collisions. However, when we reach very high - less robust - modulation schemes, there is a significant drop in both metrics, similar to the *Low SNR* case.

criterion	'gini','entropy'
splitter	'best','random'
min_samples_split	2,3,4,10
max_features (Active)	'auto','sqrt','log2',4,10,14,16
max_features (Passive)	'auto','sqrt','log2',2,3

TABLE 3.1: Decision Trees sets of hyper-parameters values

criterion	'gini','entropy'
n_estimators	5,10,20,30,40
max_features (Active)	'auto','sqrt','log2',4,10,14,16
max_features (Passive)	'auto','sqrt','log2',2,3

TABLE 3.2: Random Forests sets of hyper-parameters values

kernel	'rbf','linear'
gamma	'auto',1e-3, 1e-4
C	1, 10, 100, 1000

TABLE 3.3: SVM sets of hyper-parameters values

n_neighbors	3,5,7
weights	'uniform','distance'
algorithm	'auto','brute'
p	1, 2

TABLE 3.4: K-Nearest Neighbors sets of hyper-parameters values

Capture Effect

A more general case of the *Hidden Terminal* phenomenon is the *Capture Effect*, or else asymmetric Hidden Terminal, where frames coming from the transmitters of the

evaluation and the interfering link have a noticeable difference in RSSI value. The frame with the stronger signal is often correctly decoded by the receiver after a collision, and thus the transmitter continues transmissions with a minimum length CW. On the other hand, transmitter of the weak signal is heavily affected as collisions, and subsequently frame losses, are increased. We collected 1535 samples, in a similar manner to symmetric *Hidden Terminal* pathology and data exhibited equivalent behavior for both metrics across modulation schemes, although values were lower due to the heavier impact of *Capture Effect* on the evaluation link.

In Figure 3.1 we depict the whole data set of 5348 samples, along with their corresponding pathology.

3.4 Evaluation of classification algorithms

3.4.1 Classification Algorithms

In order to classify an underlying pathology into one of the aforementioned categories in the previous subsection, we employed supervised learning and four popular classification algorithms: a) *Decision Trees*, b) *Random Forests*, c) *Support Vector Machines* Classification and d) *K-Nearest Neighbors*.

Decision Trees

Decision trees builds classification models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets, while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches and a leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor is called root node. Decision trees can handle both categorical and numerical data.

Random Forests

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.

Support Vector Machine Classification

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

	Low SNR	WiFi	Capture	Hidden	MW
Low SNR	100	0	0	0	0
WiFi	0	100	0	0	0
Capture	0	0	97.6	2.4	0
Hidden	0	0	0	100	0
MW	0	0	0	0	100

TABLE 3.5: Confusion matrix K-Nearest Neighbors - Active Model

K-Nearest Neighbors

The K-Nearest Neighbors algorithm is a classification algorithm, and it is supervised: it takes a bunch of labeled points and uses them to learn how to label other points. To label a new point, it looks at the labeled points closest to that new point (those are its nearest neighbors), and has those neighbors vote, so whichever label the most of the neighbors have is the label for the new point (the "k" is the number of neighbors it checks).

3.4.2 Data Modeling and Feature Extraction

The experimental data gathered during the sessions described in Section 3.3, was fed to classification algorithms, modeled in two distinct ways. The first model fed classifiers with samples as 3-dimensional vectors of features, in the following format:

$$\{NCA, FDR, MODULATION\}$$

along with their labels denoting the pathology, while the second model aggregated the results of *Varying Bitrate Probing* and fed them as 16-dimensional vectors of features, in the following format:

$$\{NCA_{MCS0}, FDR_{MCS0}, ..., NCA_{MCS7}, FDR_{MCS7}\}$$

along with their corresponding labels. The rationale behind this differentiation was that although the second model, called *Active Model* onwards, can certainly offer better accuracy, it requires that the active sampling test (*Varying Bitrate Probing*) is run every time we need to detect an underlying pathology, inducing saturated traffic of no value for the user in the network. However, in the first model, called *Passive Model* onwards, samples can be obtained with no need for extra probing traffic, as described in Subsection 3.3.3.

3.4.3 Classifiers Implementation

After processing data for feature extraction, we shuffled and splitted it into a training and a test set. For the first model we selected the 80% of the samples for training and the remaining 20% for testing, while for the second model the percentages were 85% and 15% respectively, where the number of instances was decreased by the feature

	Low SNR	WiFi	Capture	Hidden	MW
Low SNR	99.3	0	0.4	0	0.3
WiFi	0	100	0	0	0
Capture	1.4	0	90.9	7.4	0.3
Hidden	0	0	6.2	93.8	0
MW	0	3.1	3.1	0.9	92.9

TABLE 3.6: Confusion matrix K-Nearest Neighbors - Passive Model

aggregation. All classifiers were implemented in Python using the *scikit-learn* library [36]. We then performed hyper-parameter tuning for both models and all classifiers, in regards to optimize classification precision. We chose precision over accuracy in an effort to avoid the accuracy paradox that can often occur in unbalanced training sets. In order to avoid overfitting, we performed a 5-fold cross-validation.The hyper-parameters for each classifier along with their set of values that were tested are given in Tables 3.1, 3.2, 3.3 and 3.4 accordingly.

More specifically, for the *Decision Trees* classifier, we found that for both models "entropy" was the best function to measure the quality of a split, which uses the information gain. The best random split was found as the best strategy for the splitter, while the minimum number of samples required to split an internal node was 4. The differentiation between the two models, *Active* and *Passive* was the number of features to consider when looking for the best split, which for the former the best value was the "auto", considering the total number of features, while for the latter was the default value 3.

Regarding, *Random Forests* classifier, again, "entropy" was the best function to measure the quality of a split. For the *Active* model, the optimized value for the number of features to consider when looking for the best split was 8 and the number of estimators was the default 10. In contrast, for the *Passive* model, the corresponding values were the "auto", representing the square root of the total number of 3 features and 30 for the number of trees in the forest.

Concerning, the *SVM* classifier, the default "rbf" kernel yielded the best results for both models, while the kernel coefficient "gamma" was set to 0.001 for the *Active* model and to "auto" for the *Passive* model that equals to *1 / total number of features* and in our case equals to 0.33. Finally, penalty parameter "C" of the error term was set to 1 for the *Active* model and 10 for the *Passive*.

Lastly, the hyper-parameters of *K-Nearest Neighbors* classifier were also fine-tuned, in order to optimize precision. As regards to the *Active* model, the number of neighbors used was set to 3, while the "brute force" search was the algorithm used to compute the nearest neighbors with a "uniform" weighting function. For the *Passive* model, we set the default value for the number of neighbors equal to 5. The algorithm used was set to "auto", in order to decide the appropriate between KDTree, BallTree and brute force according to the data and used the "distance" weighting function that weights points by the inverse of their distance. We set 'p' equal to 1 that corresponds to the Manhattan distance for the power parameter for the Minkowski metric, in both models.

3.4.4 Pathology Detection Performance

Having determined the best hyper-parameters for each of the considered classifiers and for both the *Active* and *Passive Model*, we compared them in terms of accuracy, precision and recall. The overall results are shown in Table 3.7, where it is evident that the *K-Nearest Neighbors* classifier is superior to the rest for all metrics in both models. More specifically, it exhibits an outstanding accuracy of 99.2% and 95.1% for the corresponding models. This can be attributed to the fact that the *K-Nearest Neighbors* algorithm tends to perform very well in cases when there are many data points and few dimensions of the feature vector. It is also known that as a non-parametric algorithm it performs better than parametric ones (SVM), when the considered classes are overlapping. In Tables 3.5 and 3.6, the confusion matrices of the *K-Nearest Neighbors* classifier for each model are presented. At this point, it is also remarkable to note that there is some misclassification between *Hidden Terminal* and *Capture Effect* pathologies, a fact that is quite expected, as these pathologies differ only in their symmetry.

	Active			Passive		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
DT	97.8	97.8	97.8	92.8	92.8	92.8
RF	98.1	98.1	98.1	94.6	94.6	94.6
SVM	98.7	98.7	98.7	94.1	94.3	94.1
KNN	99.2	99.2	99.2	95.1	95.1	95.1

TABLE 3.7: Evaluation results

3.5 Conclusion

This chapter presented the employment, the fine-tuning and evaluation of multiple classification algorithms for diagnosing the causes of WiFi underperformance. The most effective classifier was integrated into a software framework we built for detection supporting two operational modes: a highly-accurate *Active* and a no-overhead *Passive* one. Both modes leverage MAC-layer data exported by the driver of the wireless card. We exhibited remarkable results of accuracy, with results of 99.2% for the active and 95.1% for the passive mode. A natural extension of this work would involve the exploration of multi-label classification in an attempt to accurately detect multiple coexisting pathologies, which is a scenario that often arises in dense and complex wireless network deployments.

Chapter 4

On the Implementation of a Cross-Layer SDN Architecture for 802.11 MANETs

Contents

4.1 Introduction **33**
4.2 Related Work **35**
4.3 Proposed SDN Scheme **36**
4.4 Evaluation **42**
4.5 Conclusion **45**

The adoption of the Software Defined Networking (SDN) paradigm is aggressively expanding from traditional datacenter networks to 5G and IoT deployments due to its flexibility in network management and routing. Mobile Ad-Hoc Networks (MANETs), with their unique characteristics and requirements stemming from the mobility and volatile wireless medium conditions, have not yet realized the benefits of the SDN approach. Adoption has been hampered by technical challenges and the contradiction between the centralized concept of SDN and the distributed one of traditional MANET routing schemes. In this chapter, we present our implementation of a fully-fledged SDN framework, which attempts to bridge the gap by combining the benefits of both worlds. Our developed prototype integrates with 802.11 Ad-Hoc networks and offers a) fault tolerance for the SDN controller by recovering from failures with the (re)-election of a new controller, b) dynamic network topology discovery, and c) a cross-layer approach in routing based on 802.11 MAC layer statistics that more accurately characterize link quality and capacity. We validated the applicability and performance benefits of our method by evaluating our proposed SDN architecture in a proof-of-concept scenario.

4.1 Introduction

Software Defined Networking (SDN) is an emerging, evolutionary paradigm that provides dynamic and efficient network configuration and management by decoupling the control plane from the data plane. This allows for a holistic and centralized view of the network, upon which the SDN controller makes intelligent routing and forwarding decisions. It enables the execution of Quality of Service (QoS) policies, the deployment of new, complicated and sophisticated routing protocols, and the administration of heterogeneous switching equipment without the need for additional hardware or manual configuration. SDN's flexibility in traffic management

and efficiency in network operation have encouraged both industry and academia to swiftly adopt it as the dominant networking paradigm. According to [37], the SDN market is anticipated to witness a substantial growth from USD 13.7 billion in 2020 to USD 32.7 billion by 2025, at a Compound Annual Growth Rate (CAGR) of 19% during this period.

However, its adoption in wireless networks and, more specifically, in Mobile Ad Hoc Networks (MANETs) is not prevalent due to a number of unresolved research challenges, despite extensive academic study. The majority of obstacles originate from the dynamic and volatile wireless environment caused by the unpredictable mobility, arrival, and departure of wireless nodes, as well as the inherent conflict between the centralized structure of SDN networks and the distributed nature of MANETs. The SDN controller, the central entity, is susceptible to failures or disconnections that render the entire network inoperable. In addition, the dynamic movement of network nodes makes establishing connectivity channels between them and the controller a difficult puzzle to solve. These challenges go beyond the realm of research and are manifested in the form of technical difficulties and trade-offs that must be addressed for delivering a usable and efficient solution.

Traditional MANET routing protocols, such as Optimized Link State Routing Protocol (OLSR) [38], are able to adapt to dynamic network environment changes, exhibiting an acceptable performance, by leveraging their distributed nature to discover neighboring nodes and forward network traffic. However, the benefits of such protocols are accompanied by higher network overhead due to message exchanges and lengthy convergence times, which hinder performance in low-capacity and sparse networks, as well as in scaled and complex mesh networks. Their primary limitation, though, is the lack of a global network perspective with data coming from the various layers of the network stack, as well as the inability to act on this data with the application of intelligent policies, due to their lack of programmability.

This chapter presents and evaluates our design and implementation of a fully-fledged SDN framework for MANET networks, which attempts to bridge the gap between distributed MANET protocols and centralized SDN approaches by adopting their strengths and overcoming their constraints. Specifically, our contributions can be summed up as:

- Definition and development of an in-band SDN architecture in which the connection to the controller, part of the MANET, is also SDN-based.
- Provision of fault-tolerance and resilience in the event of controller failure or out-of-range movement, with automatic (re-)election of a new controller.
- Development of a distributed, OLSR-like topology discovery mechanism that permits dynamic network composition and quick reaction to network changes.
- Integration with the wireless 802.11 protocol via tunneling and the development of agents that extract MAC/PHY layer statistics from the wireless card driver.
- Implementation of an adaptive forwarding process that is aware of the wireless network's utilization.

This is, to the best of our knowledge, the first attempt to develop a fully operational SDN framework that addresses the majority of MANET challenges and may be regarded as a competitive alternative to routing protocols such as OLSR.

The remaining sections of this chapter are structured as follows: The section 4.2 provides a summary of relevant earlier work on SDN approaches for MANETs. Section 4.3 describes in detail each of our contributions while presenting the overall architecture of our developed framework. In section 4.4, a proof-of-concept use case deployed on a wireless testbed and a mobility scenario deployed in an emulation environment are used to evaluate the framework. Section 4.5, concludes the chapter by reviewing our proposed scheme and findings and identifying future research directions.

4.2 Related Work

Originally, the SDN concept has emerged for providing flexibility and dynamic management for the centralized networks of datacenters and mobile networks' backhauls. Nevertheless, several approaches have explored the integration of SDN with wireless networks, particularly wireless mesh networks. Prior work in [39] provides a hybrid framework that employs OpenFlow to route data traffic, while it uses OLSR for routing OpenFlow control traffic and data traffic in the event of an SDN controller failure. The evaluation was conducted within an emulation environment. Additionally, the work in [40] represents a deployment that allows for automatic selection of an SDN controller through an election procedure that then imposes routing rules with a higher priority than those of the existing routing protocol (Babel) used for providing connectivity between the nodes.

The authors in [41, 42] propose a flexible SDN architecture in which nodes use backup paths discovered in a distributed manner, when there is detection of failure of the primary path installed by the SDN controller. Although this approach can provide quick reaction to network changes, it is not able to leverage a centralized network overview for performing optimal routing decisions. Another SDN-to-MANET integration effort is presented in [43] that exhibits quick reaction times to network volatility. However, the approach is based on the assumption that the nodes possess two wireless interfaces, with the one being used for the communication with the controller, which is assumed to be always in direct connection. The works in [44, 45] provide the foundation for QoS in SDN routing, the former by prioritizing traffic flows and the latter by considering the MANET's bandwidth utilization in taking routing decisions, although both are assuming a direct, always-available connection with a central entity responsible for managing the SDN network.

Our implementation, outlined in this chapter, distinguishes itself from prior work in that, to the best of our knowledge, it is the first to offer a fully-fledged solution that addresses all aspects and issues of the integration of SDN into MANETs. It ensures deployment applicability by establishing and developing an in-band architecture requiring a single interface for both the control and data planes. In addition, the automatic election of the SDN controller improves robustness and eliminates single points of failure. Finally, and most importantly, it follows a cross-layer approach, incorporating MAC layer statistics for the accurate estimation of link capacities and the QoS routing of flows.

4.3 Proposed SDN Scheme

In this section we describe the entire SDN framework we designed and developed, providing an overview of the architecture along with details for each distinct process and functionality.

4.3.1 Overall Design

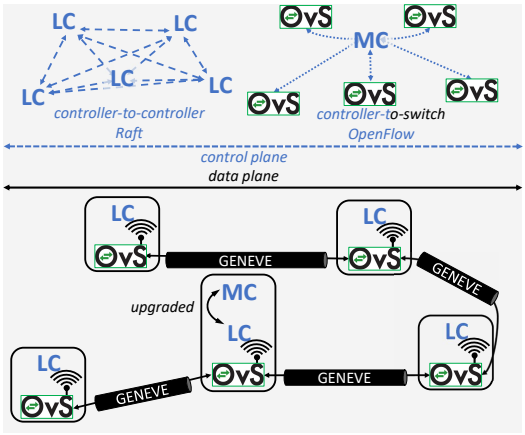


FIGURE 4.1: Overall Architecture

A high-level description of our fault-tolerant SDN framework is depicted in Figure 4.1, where each node in the MANET network is running an SDN *Local Controller* (LC) application in parallel with a virtual switch, based in Open vSwitch (OVS) [46]. In general, in the SDN concept there are two options for the implementation of the control plane, which is the means of connection between the controller(s) and the switches. The first, the out-of-band control plane, connects the controller(s) to the switches via a separate network. The second, the in-band control plane, utilizes the same network for both control and data traffic. Although it is technically challenging, we opted for the second option as it offers us the ability to completely control the network in an SDN manner and it is applicable to devices with a single wireless interface. We provide more details on how we handle the SDN control plane establishment with the assistance of LCs and the routing of control messages in subsection 4.3.5.

Furthermore, in order to ensure reliability and fault tolerance for our system, we have employed the Raft Consensus Algorithm [47] as the distributed protocol for the coordination of controller-to-controller communication between LCs and the election of a suitable *Master Controller* (MC), which is the node responsible for managing the SDN network of a MANET cluster and applying flow rules to the virtual switches, via OpenFlow, the de-facto controller-to-switch communication protocol. Ultimately, the entire SDN architecture resides on top of an 802.11 Ad-Hoc network, with which is seamlessly integrated to enable efficient control and intelligent routing.

4.3.2 Integration with IEEE 802.11

For the deployment of our SDN framework, the integration with the underlying wireless communication protocol was necessary. Although the framework is portable and directly applicable to any wireless infrastructure, we focused on 802.11 protocol (WiFi), as it is the prevalent in MANETs. In our setup, the wireless nodes are configured as Ad-Hoc (IBSS), capable of communicating directly with their one-hop neighbors. As described in [48], IEEE 802.11 uses a different addressing scheme than the traditional 802.3 standard, making integration with a virtual switch, specifically with OVS, more challenging. Currently, only IEEE 802.11 interfaces configured in 4-address mode can be directly connected to OVS. However, this mode cannot be configured when the wireless interface is in Ad-Hoc mode, as the third address of the MAC header is reserved for use as the BSSID address by the standard. To overcome this challenge, we used a strategy similar to that described in [49]. We employed GENEVE tunnels to create a virtual point-to-point link between two nodes and attached them as ports in OVS, thus making the use of IEEE 802.11 completely transparent for the virtual switch.

Besides interfacing with OVS, our cross-layer SDN architecture integrates with the 802.11 MAC layer for performing routing of traffic by leveraging MAC/PHY layer statistics that are exported by the drivers of the wireless cards such as the Packet Delivery Ratio (PDR), the TX PHY bitrate as well as the airtime utilization. The SDN controller is able, through our implemented extension of OVS and a standard OpenFlow procedure, to centrally collect them and gain an accurate estimation of the quality of the wireless links between nodes. Subsection 4.3.6 offer a more thorough analysis.

4.3.3 Fault Tolerance

The SDN Controller is the heart of the SDN network and the most critical component. As such, it is the network's single point of failure. In a MANET environment, where the Controller is mobile and moves out of the range of the switches/nodes, the network becomes dysfunctional, unable to establish new communication paths, adapt to changes in the wireless environment, or maintain centralized decision control in general. To address this challenge, we designed and developed a fault-tolerant SDN network in which each node can immediately assume the role of the MC when necessary. To do this, each node in the network, as mentioned earlier, runs an LC process concurrently with an OVS instance, which is configured to connect with all other LCs/nodes in the network. By exploiting OpenFlow's capability of assigning different responsibilities to controllers, we are able to avoid conflicts caused by concurrent and potentially contradictory flow entry installations in switches and coordinate the coexistence of numerous controllers appropriately. According to the OpenFlow specification 1.3.1 [50], a controller can operate in one of the following roles:

- Equal: This is the controller's default role. The controller has complete access to the switch and is equivalent to other controllers in the same role.
- Slave: A Slave controller has access to the switch in a read-only mode. The controller is prohibited from issuing controller-to-switch commands.

- Master: The Master controller has complete read/write access to the switch. At any given time, only one controller can be the Master.

Thus, while every switch in the network runs an LC process, only a single of them can be upgraded to the MC role at any given moment, able to control and apply flow rules to the rest, therefore retaining SDN's centralized nature. This controller is configured as the Master in the OvS instances of the network. The rest are configured as Slaves for all OVS instances apart the ones that are collocated, for which they are configured as Equals.

This strategy eliminates the single point of failure providing robustness and resilience to the network. In our implementation, the employment of Raft election scheme is exploited for the nomination of the MC by automatically declaring the single *leader*, in Raft terminology, of a cluster of nodes as the MC of the SDN network. Specifically, the controller of the Raft-elected leader node communicates its intention to become MC with the use of OpenFlow *Role Request* messages.

There are two parameters that are controlling the behavior of the Raft algorithm, and therefore the controller-to-controller communication of the SDN network: a) the *heartbeat interval* and b) the *election timeout*. Heartbeat interval (*hbeat_interval*) defines the frequency with which the leader will notify followers that it is still the leader. Election timeout (*election_timeout*) is how long a follower node will go without hearing a heartbeat before attempting to become leader itself. Both should be tuned appropriately so that leadership is transferred rapidly in case of leader failure or disconnection and at the same time unnecessary elections and extra overhead are avoided.

4.3.4 Topology Discovery

Every MANET protocol's core functionality is neighbor discovery, as any node must be aware of its one-hop neighbors in order to establish multi-hop connectivity with any other node in the network. This is no different in our SDN case. However, due to the centralized nature of the SDN network, the controller, and specifically in our implementation the MC must be aware of the entire network topology. A common procedure in SDN wired networks, but also in wireless [51], for topology discovery, is through centralized circulation of LLDP packets between the controller and the switches and the subsequent building of a network graph.

This centralized approach comes with two major disadvantages. The first is that an LLDP packet originating from an SDN controller may traverse the entire MANET network to arrive back to the controller as an OpenFlow *Packet In* message, for the discovery of just a single link. Accounting the fact that this process must be repeated for every node and every port of the node's virtual switch, periodically, it is evident that the overhead in the network grows significantly as the size of the MANET network grows. The second is that having only a single SDN controller aware of the network's topology (i.e. MC for our case), there is a risk of losing connectivity during elections or during bootstrapping of the system. In order to overcome these challenges, we implemented a more distributed approach, similar to the one of OLSR.

In specific, the collocated LC of each node, operating in Equal mode, instructs the switch to broadcast special messages through the broadcast GENEVE tunnel we use

to avoid overhead of forwarding broadcast messages through individual tunnels. Such messages include the *Hello* messages that are used for discovering neighboring links, and *Topology Control* (TC) messages that are used to discover multi-hop paths, both encapsulated in an LLDP packet as separate sections. The Hello message section of the LLDP packet consists of the originator node identifier and a bit that indicates if the message has been rebroadcasted. The TC message section contains the originator node's incoming one-hop links which are learned from previous Hello messages received.

The LLDP packets containing the Hello and TC sections are broadcasted every *lldp_period* ms and are rebroadcasted by all nodes using sequence numbers in order to avoid loops. Every node has a local topology database where it stores the links that learns through LLDP exchanges. In case a new TC message arrives that it does not contain a previously stored link, then this link is deleted from the database. Accordingly, the record of a neighboring link in the database from where the node does not receive a Hello message within *timeout_period* seconds is also deleted. For non-neighboring links, if no TC message is received after timeout_period seconds from the last TC message received from the same originator, then all links stored in the database associated with this originator are deleted as well. Therefore, with this distributed approach, every node and its respective collocated LC is able to construct a network graph with the discovered nodes (neighboring or not) allowing network connectivity during network bootstrap or MC election.

4.3.5 Flow Entries Management

Our SDN framework must effectively handle a number of difficult special circumstances that arise throughout the operation of the network lifecycle. First and foremost is the management of the network bootstrap that will allow the discovery and connection between the nodes and the non-collocated LCs. As was already mentioned, the switch's collocated LC manages the switch in tandem with the MC, when operating in Equal mode. The LC uses the network graph learned from the topology discovery to setup flows in its collocated switch to route control traffic. These flows are also used to route data plane traffic when an MC has not yet been elected, or if the LC has not been connected (or lost its connection) to the elected MC, to service data traffic in the absence of an MC. As a result, even if they only handle their switch, the collocated LCs are able to create a network graph and coordinate the forwarding of packets, both control and data, to the discovered network using the discovery mechanism described in the preceding subsection. This process is carried out concurrently by all LCs during network startup, and once the absolute majority of the cluster has been discovered and connected, the election of an MC can be completed and the elected controller will then take control of managing data flows for the whole network.

The controller, either operating as an MC or collocated LC, instructs the switches to forward packets by installing flow rules based on the Dijkstra shortest path found on the graph that has built and annotated with appropriate weights. The process of calculating the weights will be analysed in the following subsection. Each flow rule installed has an *idle_timeout*, meaning that if the flow remains inactive for a period larger than the timeout, the flow is deleted and the controller is notified. This allows switches to inquire for new (perhaps better) paths, when the flow is activated again. Furthermore, the installed flow rules on the switches are also stored internally by

the controller process, in order to be able to manage them in case the graph is altered by a network event such as a new link discovered or an existing link removed. In more detail, a separate thread of the process, periodically inspects the stored flow entries and queries the graph for better paths, and when one is found, new rules are installed on the involved switches via *Flow Mod* messages. This enables data traffic to be always forwarded through the links providing the best performance. Both idle_timeout and period of updating the flows are apt to judgement and configuration of the network administrator, as the associated performance benefits are dependent to the volatility and size of the network.

Furthermore, when an MC, during Packet In handling, is unable to retrieve a path towards a destination, then it removes the respective flow entry for this destination at the switch of the source node that generated the data packet (if it exists). This last condition refers to the case, when a packet leaves its source and arrives at an intermediate node, between source and destination, which inquires the MC via a Packet In message. In this case, where the MC is unable to forward the packet, the flow entry at the source should be removed, otherwise the packets will keep following a path with dead end.

4.3.6 Cross-layer SDN-MAC integration

A significant advantage that an SDN network can provide us over the use of traditional MANET routing protocols, is the ability of the SDN Controller to perform intelligent routing decisions based on the knowledge that collects and stores from across the network nodes and between the various layers of the network stack. To that end, we have enabled the integration between the SDN control plane and the 802.11 MAC layer, by implementing the mechanisms that allow the controller to acquire data and statistics from the open-source driver of the wireless card of each node, following a similar approach with our previous works [1],[2]. These statistics are leveraged by the controller in order to define the weights, for the links in the graph, which are accurately characterizing the links' available capacity. These weights are calculated every τ period using the equations below, where r is the median of the PHY rates reported by the driver for the same period.

In the CSMA/CA based MAC layer of 802.11 where multiple stations are competing for the access to the wireless medium, there are several factors that influence the actual throughput performance of a transmitting station and these are also related with the assesment of a link's available capacity. For a time interval $[t_0, t_0 + \tau]$ the available capacity $AC(t_0, \tau)$ of a link between a source node s and the directly connected destination node d is estimated as follows

$$AC_{sd}(t_0, \tau) = \frac{1}{\tau} \int_{t_0}^{t_0+\tau} \text{PDR}(t) \times (1 - u(t)) \times C(r) dt. \tag{4.1}$$

From Equation 4.1 we can observe that the available capacity of a link for a specific time interval depends on the calculated Packet Delivery Ratio PDR(t), the fraction of time $u(t)$ during which the medium is sensed as busy (airtime utilization) and the maximum link capacity $C(r)$. It is worth to note here, that $u(t)$ refers not only to the proportion of time required for the transmission of the traffic of the network's nodes, but also accounts for the interference from external networks or devices. The $C(r)$ for an 802.11 link (n version and above), assuming no losses and a simplified

model based on the analysis in [52], is in turn calculated as follows

$$C(r) = \frac{\text{maxAMPDU}(r) \times L}{\text{TxDelay}(r)}, \tag{4.2}$$

where maxAMPDU(r) is the maximum number of frames allowed into an AMPDU frame for a PHY rate r, and L refers to the UDP payload carried by a single IP packet. The TxDelay corresponds to the transmission delay of the AMPDU frame for the PHY rate in use, which also involves the T_{WIFI} delay caused by management and control frames along with the backoff delay, all associated with the standard process of an 802.11 transmission, and which equals $T_{\text{WIFI}} = 249\mu s$, when CTS, RTS and ACK are transmitted with 6 Mbits/s PHY rate

$$\text{TxDelay}(r) = T_{\text{WIFI}} + T_{\text{DATA}}(r), \tag{4.3}$$

where $T_{\text{DATA}}(r)$ can be approximately defined as

$$T_{\text{DATA}}(r) = T_{\text{PH}} + \frac{\text{maxAMPDU}(r) \times (S_{\text{MAC}} + S)}{r}, \tag{4.4}$$

where $T_{\text{PH}} = 20\mu s$ is the transmission delay of the PHY header, S_{MAC} is the size of the MAC header and S is the size of a single frame.

In order to avoid unnecessary complexity by introducing bi-directional edges in the graph, we decided to denote as the available capacity for a link, the minimum between those estimated for each of the link's end points. Therefore, we annotate each edge between two directly connected nodes, s and d in the graph, with a weight w_{sd}, calculated as

$$w_{sd} = \frac{1}{\min(AC_{sd}, AC_{ds})} \tag{4.5}$$

Consequently, when a Packet In for a new flow arrives, the controller performs shortest path computation in order to establish the forwarding path of the flow. However, during the process of periodically updating the flows, the controller inspects every link's available capacity in order to avoid congested links. In such links, where AC is under a pre-defined threshold thr, the controller rearranges the flows, starting with the one, f, with the less consumed bandwidth $bw_{f,sd}(t_0, \tau)$, in order to bring AC over the thr again. For each flow f and for the given time interval $[t_0, t_0 + \tau]$, the consuming bandwidth for a specific link $s \rightarrow d$ is approximated by

$$bw_{f,sd}(t_0, \tau) = \frac{\text{TxBytes}_f(t_0, \tau)}{r}, \tag{4.6}$$

where $\text{TxBytes}_f(t_0, \tau)$ are the transmitted bytes for flow f as reported by the switch. The controller, thus, selects the alternative path that minimizes the aggregated link congestion (closely related to the weight w), of the whole network.

The aforementioned parameters for the estimation of links' available capacities and therefore the edges' weights, as well as consumed bandwidths, are polled and collected from the open-source driver of the wireless card by our implemented agent, which is an extension of OvS, responsible for communicating these statistics to the MC and to the collocated LC. The interval, over which the agent inquires the driver is called l2_sample_period and can be configured by the network administrator. Measuring PDR and airtime utilization for the specified interval is straightforward. The

TABLE 4.1: Experiment Parameters

Parameter	Value
lldp_period	2 s
timeout_period	20 s
hbeat_interval	100 ms
election_timeout	1000 ms
l2_sample_period	5 s

exchange of the statistics is still OpenFlow compatible as we are utilizing the OpenFlow *Experimenter* messages that are defined by the standard and which allow us to pass arbitrary data. Furthermore, the actual transmitted bytes for each flow that are reported by the switches are encapsulated in OpenFlow *Flow Stats* messages.

4.4 Evaluation

In order to test and evaluate our developed framework, we deployed it on a realistic testbed environment and assessed its performance under a particular, but commonly encountered scenario. We focused on routing decisions in the presence of utilized links, which allowed us to further illustrate the rationale and performance benefits of our cross-layer SDN approach.

4.4.1 Testbed Setup

Our testbed comprises of six wireless nodes, which are essentially commercial off-the-shelf computers running Ubuntu 18.04 and equipped with an 802.11ac-capable wireless card. The card's chipset is QCA6174, and the corresponding open-source driver is ath10k. In addition, we have installed our framework, which consists of numerous software modules and scripts. We have specifically deployed our extended version of OVS, based on version 2.15.90, our expanded version of Ryu Controller [53], based on version 4.34, our developed SDN application that implements the controller architecture mentioned in section 4.3, and etcd [54], as a distributed storage and implementation of Raft, version 3.6. Several scripts have been developed to automate the configuration and initialization of the wireless interface, the GENEVE tunnels, and the virtual switch. These scripts are executed as startup services.

As is obvious from the preceding section, there are a number of configurable system parameters that can considerably impact its performance. a) the lldp_period, b) the timeout_period, c) the hbeat_interval, d) the election_timeout and e) the l2_sample_period are the most notable. The values of these parameters are closely related to the characteristics of the network, such as its size and mobility of nodes, and they create a trade-off between rapid adaptation and network overhead. However, their optimal configuration is studied in the next chapter, so we have specified the values that we deem most suitable for our case, which may be found in Table 4.1.

In order to evaluate our SDN framework, the network would have to feature several multi-hop paths between pairs of nodes so that the controller's routing decisions could have an impact. In order to achieve this multi-hop architecture in our lab's restricted area, we attached multiple RF signal attenuators to the wireless interfaces

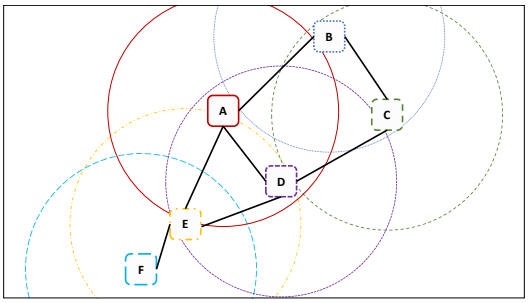


FIGURE 4.2: Network Topology

TABLE 4.2: Observed Metric Values at t=0s

Link	PHY	PDR	Ut	w	DAT
A-B	MCS 5	0.9	0.27	0.026	52.02 Mbits/s
A-D	MCS 7	1	0.27	0.019	72.2 Mbits/s
B-C	MCS 7	0.78	0.12	0.020	56.32 Mbits/s
D-C	MCS 5	0.91	0.25	0.025	52.60 Mbits/s

and configured them to use a single antenna, therefore reducing signal quality without physically distancing the nodes. The nodes were eventually arranged in the static topology represented in Figure 4.2. The wireless cards of the nodes were configured in Ad-Hoc mode, meaning that the nodes had connectivity with the one-hop neighbors, but there was no routing algorithm by default to allow multi-hop connections.

4.4.2 Contending Link Experiment

In this proof-of-concept scenario, we aim at investigating the routing decision performed by the MC for establishing a multi-hop path for a flow between a pair of nodes, in the case where a contending flow has been activated between nodes that are not directly involved in the path. All nodes are configured to operate in a 20 MHz channel. We compare the decision of our SDN framework with the one of the widely adopted OLSRv2. OLSRv2 uses the Directional Airtime Metric (DAT) [55] in order to characterize the quality of the established links. In short, this metric is a successor of the ETX metric and essentially takes into account the link speed, via an external measurement process and the packet loss rate through analysis of multicast control traffic. DAT can be expressed with a human readable value of link speed in bits/s, between 119 bit/s and 2 Gbit/s.

We conducted two experiments, one with the SDN framework and one with the OLSR, both under the same scenario. We start the scenario with the activation of a UDP flow between nodes A and C. The data rate of the flow is 30 Mbit/s and is enough to saturate both the two-hop links between A and C. The first refers to the path through relay node B and the second through relay node D. In Table 4.2 we show the observed metrics as collected by the respective modules at *t=0s*. Although the network is essentially idle, the airtime utilization values are considerably over zero, as they represent the time for the transmission of SDN/OLSR control messages as well as 802.11 control and management frames. However, the link between B and C exhibits lower utilization due to the nodes being more isolated. By inspecting our

TABLE 4.3: Observed Metric Values at t=15s

Link	PHY	PDR	Ut	w	DAT
A-B	MCS 5	0.82	0.38	0.034	49.13 Mbits/s
A-D	MCS 7	0.98	0.38	0.023	70.03 Mbits/s
B-C	MCS 7	0.72	0.14	0.022	50.54 Mbits/s
D-C	MCS 5	0.8	0.4	0.036	46.24 Mbits/s

defined weights (w) and the DAT values is evident that both SDN and OLSR will prefer to route the flow through node D and this is exactly what happened. Figure 4.3 shows the achieved throughput of the flow for the period that it was active for both SDN and OLSR. In *t=10s* we stop the flow and we activate a new flow of 10 Mbit/s UDP traffic between node E and F, which are directly connected. After 5s, in *t=15s*, which is enough time for the metrics' values to be updated, we activate again the flow between A and C. The observed metric values are referenced in Table 4.3.

The inspection of the table allows us to derive some very useful insights. We observe that the utilization values are significantly increased for both nodes A and D, while for node B it remains quite similar with the case where the flow between A and C was the only one existing in the network. The transmission PHY rate as well as the PDR remain at similar values for all the links involved. This can be attributed to the fact that node E is in the sensing range for both nodes A and D, resulting in both nodes spending significant time sensing the medium busy. However, as both nodes are also in the range of node E, when it is their turn to transmit, E will sense the medium busy and defer its transmission. This explains why PDR remains unaffected, as the carrier sense of 802.11 CSMA/CA guarantees a minimum, close to zero, amount of collisions and consequently, the rate control algorithm keeps the same selection of PHY rate. This is the same reason why OLSR that considers only link speed and packet losses, calculates similar values for the DAT metric, as it does not account for the utilization of the links. Observing the weights we can deduce that the SDN controller selects the path through node B, while OLSR does not change its selection of a path through node D. Eventually, Figure 4.3 shows the achieved throughput for the last and the previous stages of the scenario for both SDN and OLSR. A major throughput improvement, around 41%, in this scenario, is evident when routing accounts for the utilization of the wireless medium, as in the case of our SDN framework.

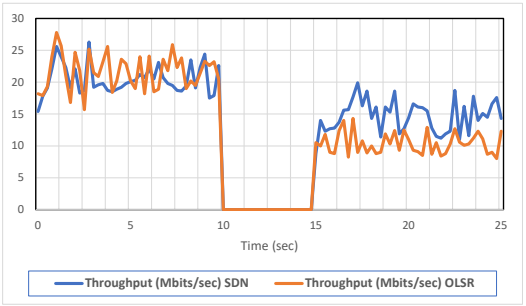


FIGURE 4.3: Throughput of flow between A and C.

4.5 Conclusion

This chapter has presented the implementation of our SDN framework for an 802.11 MANET network that effectively adopts and combines the strengths of traditional MANET routing protocols and SDN architectures. Our developed solution is fault-tolerant and resilient with each node ready to assume the role of the SDN controller for the whole network, through a Raft-based election process. It features a distributed topology discovery process and is based on an in-band architecture allowing the applicability on devices with a single wireless interface. Furthermore, it follows a cross-layer approach in routing by leveraging 802.11 MAC layer statistics in order to accurately capture link capacities. The benefits of such an approach have been demonstrated under a proof-of-concept experimentation.

Potential future extension of this work could include the incorporation of hierarchy and multiple tiers of SDN networks that will enable our system to scale efficiently.

Chapter 5

Dynamic SDN Configuration in MANETs: A Reinforcement Learning Approach

Contents

5.1 Introduction **48**
5.2 Related Work **49**
5.3 System Overview **50**
5.4 Behavior of System Parameters and Optimization Trade-offs . . . **53**
5.5 Self-Adaptive SDN Configuration **59**
5.6 Implementation **64**
5.7 Evaluation **67**
5.8 Conclusion **70**

The adoption of Software-Defined Networking (SDN) in Mobile Ad Hoc Networks (MANETs) is hindered by the highly dynamic links and the limited wireless resources, despite the promise of flexible, centralized control of communication infrastructures. To this end, we previously introduced a cross-layer in-band SDN framework that was designed to address all associated challenges, including topology discovery, SDN controller fault tolerance, and optimized routing, by integrating a variety of control-plane mechanisms. These mechanisms are controlled by tunable parameters such as heartbeat intervals, LLDP discovery rates and Raft election timeouts, which incur a trade-off between adaptability to changing conditions and overhead that static or manually tuned settings cannot resolve across varying topologies and network conditions. This chapter introduces an autonomous, Reinforcement Learning (RL)-based framework that continuously adjusts these critical SDN parameters in response to the real-time network conditions. Our prototype integrates into our prior SDN architecture RL agents, which leverage cross-layer network statistics to perform optimized self-configuration. We evaluated the performance of our RL-based scheme in comparison to random and static configuration strategies. Our results demonstrated that our scheme is more responsive to physical layer variations, as evidenced by the nearly 60% reduction in connectivity deviation (i.e. the gap between IP-layer and physical-layer connectivity), and exhibits better performance, resulting in 20-46% increases in throughput over the static baselines. These findings illustrate that model-free RL can dynamically balance control-plane overhead and resilience, rendering SDN viable for highly dynamic MANETs.

5.1 Introduction

The emergence of Software Defined Networking (SDN) has facilitated the provision of centralized control, dynamic and efficient network management and administration, as well as the ability to (re-)configure network components in real-time. The decoupling of the network's control and data planes allows for a comprehensive network overview that supports the implementation of quality of service (QoS) policies and intelligent routing decisions. The central component of the SDN paradigm is the SDN controller, which performs the task of consolidating and abstracting various conventional network functions. This is achieved through its interaction with network devices, specifically SDN switches. Hence, it is strategically positioned on the critical path of the SDN network to ensure the smooth and uninterrupted operation of the system.

Extensive research and academic studies have been conducted to investigate the implementation of an SDN scheme in a Mobile Ad Hoc Network (MANET). However, this adoption presents notable challenges due to the dynamic nature of wireless network environments and the unpredictable movement of nodes. These challenges result in vulnerabilities in centralized control, leading to reliability issues and frequent disruptions in connectivity with the SDN controller. Our earlier work [56] attempted to address these challenges by developing an SDN framework for MANETs. This framework incorporated various components, including a fault-tolerant mechanism to handle controller failures and a topology discovery mechanism inspired by the Optimized Link State Routing (OLSR) protocol to adapt to the dynamic nature of the network. The implementation of multiple SDN controllers in a read-only, slave mode guarantees fault-tolerance by enabling them to seamlessly take over as the Master Controller in the event of failure or when the current controller goes out of range. The Raft algorithm provides the synchronization protocol that also includes an election mechanism for determining the next master. Topology discovery is achieved by the exchange of Link Layer Discovery Protocol (LLDP) messages between the nodes and their subsequent collection by the controllers. This process leads to the creation of a network graph, which serves the SDN controllers in making efficient routing decisions.

Nevertheless, these mechanisms exhibit an inherent trade-off between adaptability and performance. The transmission of an excessive quantity of control messages, which enables rapid adjustment, imposes a substantial burden on the wireless network. This is particularly problematic as these networks are frequently utilized for mission-critical applications in tactical deployments and possess limited computing and transmission capacities, thereby their performance can be severely impacted.

In the work presented in this chapter, our objective is to achieve intelligent and automated balancing of the aforementioned trade-off. To accomplish this, we propose the utilization of Reinforcement Learning (RL) agents that fine-tune system parameters that control the exchange of synchronization and discovery messages based on the underlying network conditions and the dynamic movement of the nodes. The adoption of the RL approach is motivated by the ample availability of network data that can be gathered by the SDN controllers through the utilization of the standardized messages outlined in the OpenFlow protocol. Additionally, the absence of analytical models capable of precisely characterizing this trade-off, particularly in terms of formulating topology discovery and service continuity as a function of control message exchange frequency, further supports the encouragement of the RL

approach.

Our work's contributions can be summarized as follows:

- Extensive analysis of reliability versus performance trade-offs and assessment of the performance impact of the tunable system parameters.
- Training and deployment of RL agents capable of automatically configuring system parameters based on the observation of network monitoring data on nodes of the SDN-MANET network.
- Evaluation of our prototype implementation in a realistic tactical scenario.

To the best of our current knowledge, this research effort represents the first attempt to create an automated framework capable of promptly adjusting to the inherent network conditions and mobility patterns within a MANET. This is achieved through the refinement of system parameters to sustain an acceptable level of performance in terms of throughput, latency and packet loss.

The remainder of the chapter is organized as follows. Section 5.2 provides a summary of relevant earlier work for integrating SDN into MANETs and on the application of RL on SDN approaches. Section 5.3 describes the overall architecture of the SDN framework for MANETs and section 5.4 provides the analysis of the system parameters. In Section 5.5, we provide the details of our RL framework for self-adaptive system configuration, while in Section 5.6 the implementation specifics are given. Section 5.7 presents the results of the evaluation of our framework in realistic emulation scenarios, and Section 5.8, concludes the chapter by reviewing our proposed scheme.

5.2 Related Work

5.2.1 SDN-MANET Integration

The need for centralized management and dynamic configuration has recently emerged for inherently distributed MANETs, with a number of papers investigating SDN approaches for replacing or supplementing traditional MANET protocols [57]. Work presented in [39] describes a hybrid approach with SDN routing in the data plane and a transition to OLSR for the control plane in the case of SDN controller failure. Labraoui et al. [40] propose an architecture that includes an election mechanism for determining a master SDN controller, while connectivity is ensured by a distributed routing protocol (Babel) until a master is discovered or elected. Another SDN-to-MANET integration attempt is detailed in [43], where the rapid adaptation of SDN to network volatility is highlighted. Yet, the approach assumes that the nodes are equipped with two wireless interfaces, one for the control plane and the other for the data plane, and that the controller is always in direct connection. In another work, Streit et al. [45] investigate a Wireless SDN architecture for highly-utilized MANETs, in which an SDN controller continuously monitors per-link bandwidth usage and dynamically schedules or throttles flows to prevent congestion and preserve QoS under heavy load. However, the paper does not consider fault tolerance of the SDN Controller. While Sousa et al. [58] demonstrate that opportunistic coexistence of SDN-capable and legacy nodes can maintain or improve delivery

ratio under light load, their hybrid architecture relies on a static OLSR fallback and does not adapt any SDN control-plane parameters in real time as network conditions evolve. Dusia and Sethi [59] propose an infrastructure-less SDN-MANET architecture that operates without dedicated out-of-band channels, although their design does not include any mechanism for detecting, tolerating, or recovering from SDN controller failures. Consequently, our previous work [56] marks the first attempt to address the totality of the challenges and opportunities of the SDN-to-MANET integration by providing a fully-fledged framework that offers resilience and performance via a cross-layer approach.

5.2.2 Reinforcement Learning in SDN

The application of Reinforcement Learning and Deep Reinforcement Learning in several research fields and use-cases has garnered substantial interest from academia and industry as a means to approach decision-making and problem solving. Both have been applied in the context of communications and networking [60] to provide a solution, with the assistance of the SDN's centralised management, to traditional challenges such as routing [61], [62], rate control, caching, offloading, security, traffic engineering, resource scheduling and monitoring. Authors in [63] develop an RL-based intelligent routing protocol that considers link-state information collected via the monitoring capabilities of SDN, whereas in [64], the authors apply RL to select critical flows and then reroute them to achieve load-balancing. Lin et al. [65] propose a QoS-aware routing scheme based on RL that forwards traffic flows in a hierarchical SDN deployment while preserving the QoS requirements of applications. Additional research in [66] introduces a DRL routing algorithm using a reward function that takes into account network throughput and delay and displays superior performance compared to conventional routing protocols such as OSPF.

Reinforcement Learning has been also applied in SDN networks to regulate parameters that influence synchronization between multiple controllers, as well as network traffic monitoring. Zhang et al. in [67, 64] formulate the multi-controller synchronization problem as a Markov Decision Process (MDP) and apply DRL yielding significant performance benefits in terms of request latency compared to the ONOS default synchronization policy. In [68], the authors use a similar approach to ours, but in a different context, by applying RL in an SDN monitoring framework in an effort to balance the trade-off between high monitoring accuracy and excessive overhead.

Our work is unique in that it is the first to develop and implement an RL-based framework for the automatic and dynamic adaptation of SDN parameters to ensure the performance and resilience of a MANET deployment, setting it apart from previous research. It is highly relevant for tactical and highly dynamic environments, where network topology changes frequently due to node mobility, as it prioritizes self-learning and adaptability and bases its decisions on real-time conditions.

5.3 System Overview

In this section, we provide a concise and yet comprehensive overview of the SDN framework designed in [56], focusing on the mechanisms developed to provide synchronization and ensure the availability of an SDN controller, as well as to discover

existing or consider as non-existent links between nodes.

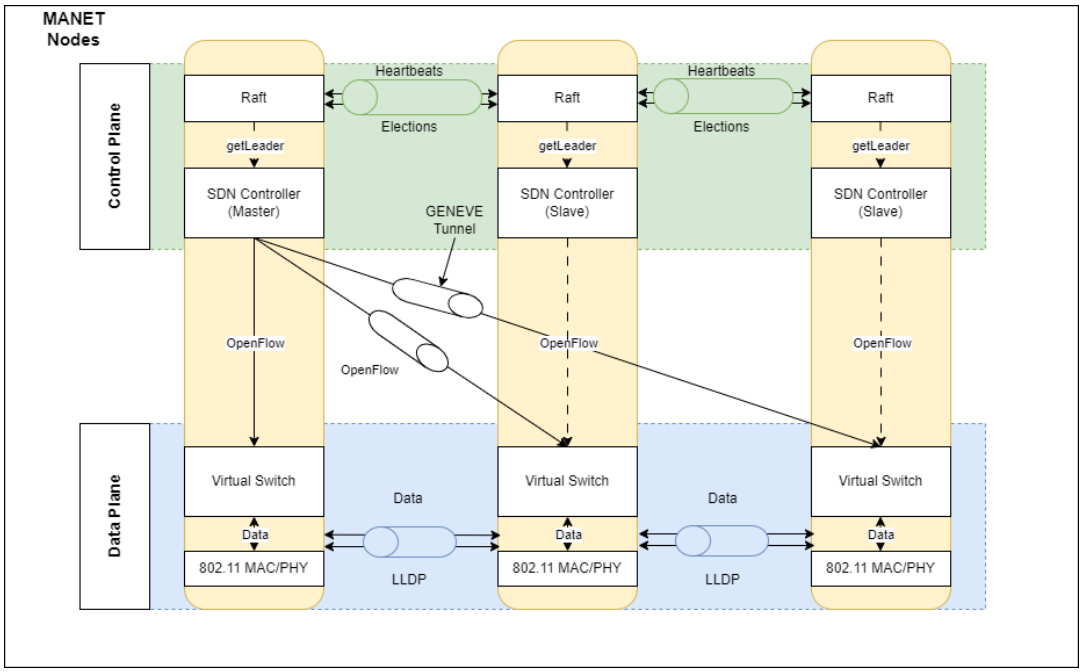


FIGURE 5.1: Architectural design of the in-band SDN framework for MANETs.

5.3.1 Overall Design

Figure 5.1 depicts a high-level depiction of our fault-tolerant, in-band SDN framework, in which each node in the MANET network concurrently runs an SDN Local Controller (LC) application and an Open vSwitch-based virtual switch (OVS). While our SDN framework is portable across any wireless infrastructure, in this work we deployed it over IEEE 802.11 ad-hoc (IBSS) networks—where direct one-hop communication and the lack of native four-address support in Ad-Hoc mode require tunneling each WiFi link into OVS via GENEVE for transparent integration.

The Raft Consensus Algorithm [47] was chosen as the distributed protocol for the coordination of controller-to-controller communication between LCs and the election of a suitable Master Controller (MC), which is the node responsible for managing the SDN network of a MANET cluster and applying flow rules to the virtual switches via OpenFlow, the de facto controller-to-switch communication protocol. The complete SDN architecture ultimately lives on top of an 802.11 Ad-Hoc network, which is seamlessly integrated to enable efficient control and intelligent routing.

5.3.2 Fault Tolerance

The SDN controller is the SDN network's core component, and at the same time, the single point of failure. In a MANET context, when the controller is no longer within range of the switches/nodes, the network cannot develop new communication paths, react to wireless environment changes, or maintain a centralized decision control. To address this issue, we developed a fault-tolerant SDN network where

each node can instantly become the MC. To achieve this, each network node runs an LC process concurrently with an OVS instance, which is set to connect with all other LCs/nodes. To minimize conflicts induced by concurrent and perhaps contradictory flow entry installations in switches, we exploit OpenFlow's ability to assign controllers various roles. The OpenFlow specification 1.3.1 [50] includes a list of the following controller roles:

- Equal: This is the controller's default role. The controller has full access to the switch as other controllers in the same role.
- Slave: A Slave controller has read-only switch access. Controller-to-switch commands are prohibited.
- Master: The Master controller has full read/write access to the switch. Only one controller can be Master.

Hence, while every switch in the network runs an LC process, only one may be promoted to the role of MC, controlling and applying flow rules to the rest, keeping SDN's centralized character. This controller is configured as Master in all OvS instances in the network. The rest are Slaves for all OvS instances except the collocated ones, which are Equals. In the event of node isolation and network partitions, we maintain the collocated LC in an Equal role to ensure that the virtual switch can be managed. Our implementation uses the Raft election mechanism to automatically nominate the cluster's single leader (in Raft terminology) as the SDN network's MC. In specific, the LC of the Raft-elected leader node communicates its intention to promote to MC with the use of OpenFlow *Role Request* messages.

5.3.3 Topology Discovery

The establishment of traffic forwarding in SDN controller necessitates the knowledge of links connecting network nodes, enabling the construction of a network connectivity graph that accurately represents the underlying topology. The majority of SDN controller implementations currently utilize the OpenFlow Topology Discovery Protocol (OFDP) to construct this graph [69], [70]. OFDP stipulates that the SDN controller encapsulates an LLDP packet within a *Packet Out* message, which is then transmitted to each switch in the network. The *Packet Out* message directs the switch to transmit the LLDP packet from each of its ports. A neighboring switch at the opposite end that receives the LLDP packet, which includes the identifier of the originating switch, encapsulates it within a *Packet In* message and then forwards it to the SDN controller. The controller after receiving the LLDP packet is thus able to infer the presence of a unidirectional link between the two switches.

In our framework we employed an OLSR-like, distributed approach. Every LC in the network instructs its collocated switch to broadcast special messages, called *Hello* and *Topology Control* (TC) messages, and which are contained in LLDP packets as separate sections, in order to discover multi-hop paths. The LLDP packets are broadcasted every `lldp_period` ms and are rebroadcasted by all nodes using sequence numbers in order to avoid loops. Each node stores the links learned through LLDP exchanges in a local topology database. If a new TC message arrives without including a previously stored link, the database deletes that link. Similarly, the database

record of a neighboring link from which the node does not receive a Hello message within `timeout_period` seconds is erased. For non-neighboring links, all links in the database associated with the same originator are removed if no TC message is received after `timeout_period` seconds from the last TC message. With this distributed approach, every node and its collocated LC can form a network graph with the discovered nodes (neighboring or not), allowing network connectivity during network bootstrapping or MC elections.

5.4 Behavior of System Parameters and Optimization Trade-offs

5.4.1 Analysis of System Parameters

Within this integrated framework, there are several adjustable system parameters that have the potential to significantly influence the performance of the network. In this section, we will provide a detailed analysis, based on experimentation, on how different values of these parameters impact the behavior of the whole network and what are the factors that should be considered when adjusting them.

There are two parameters that influence the behavior of the Raft algorithm and the controller-to-controller communication of the SDN network and these are the `heartbeat_interval` and the `election_timeout`. In the Raft consensus algorithm, the nodes assume one of three distinct roles: a) leader, b) candidate, or c) follower. The leader periodically transmits heartbeat messages to the followers as a means of informing them of his continued leadership status. The interval between two consecutive heartbeats is referred to as the `heartbeat_interval`. The followers hold a timer that undergoes a reset each time they receive a heartbeat from the leader. If the follower does not receive a heartbeat before the expiration of the timer, which occurs after the designated time period known as the `election_timeout`, it will transition to the candidate status, thereby initiating an election procedure. A candidate attains leadership status upon receiving a quorum of votes, which refers to obtaining more than fifty percent of the total votes that could potentially be cast. In the event that other nodes have transitioned into candidate status, they will revert back to followers upon receiving a heartbeat from the newly elected leader.

It is apparent that both parameters must be adjusted appropriately in order to facilitate the swift transfer of leadership in the event of leader failure or disconnection, while also minimizing the occurrence of unnecessary elections and additional overhead. A commonly suggested practice [47],[54] is to set the `election_timeout` to be approximately ten times longer than the network's broadcast time. The network's broadcast time refers to the average duration required for a node to transmit a message simultaneously to all nodes within the network and receive their respective responses. It is recommended to configure the `heartbeat_interval` in close proximity to the broadcast time. However, as previous studies [71], [72] have indicated, there is a need for new approaches to adjust the aforementioned parameters. This is necessary to account for the impacts of network disruptions, including link failures, isolation, network partitions, and node overload. Considering the higher occurrence of such phenomena in the unpredictable setting of a MANET compared to a wired network, it becomes imperative to dynamically configure parameters based on the current network conditions for maximizing the availability of the SDN network.

Furthermore, the rate at which dynamic changes in network topology and links are detected is influenced by two additional parameters. These parameters are denoted as `lldp_period` and `timeout_period` respectively, and their roles are already defined in the previous subsection. Following that description, clearly, a smaller `lldp_period` results in a more rapid discovery of new links between the nodes; however, it simultaneously increases the network overhead due to a substantial volume of LLDP messages competing with other control and data packets for a portion of the limited available airtime in order to be transmitted, and in addition, depending on the devices' capabilities, it may hinder their processing capacities. Adjusting the `timeout_period`, conversely, does not introduce additional network overhead but does impact the time required for determining whether a link has ceased to exist. A lower timeout value renders the network oversensitive to changes and even if a node temporarily moves out of range, the discovery and establishment of new routing/forwarding paths may be required for the communication across the network. In contrast, higher values hinder the network's ability to respond rapidly to changes, resulting in traffic being forwarded through invalid links and consequently causing packet losses.

To demonstrate the influence of these parameters on the performance of the network, we executed a sequence of experiments wherein we varied the values of a single parameter while maintaining the values of the others constant. We, initially, assessed the following metrics: 1) the total network overhead introduced from control messages transmitted by cluster nodes; and 2) the average percentage of time in which a cluster node perceived a Master Controller as being available. However, in order to more accurately represent network performance, we opted to also assess the average one-way delay between cluster nodes, as we consider one-way delay to be the most appropriate metric for quantifying the influence of additional network overhead due to its ability to aggregate all delays resulting from the transmission of excessive control packets. These delays include queuing, back-off, and re-transmission delays originating from the underlying 802.11 MAC layer. In order to calculate it, we assumed that every node within the MANET is equipped with a GPS module, which provides precise clock synchronization via the decoding of time signals broadcasted by GPS satellites. Based on this assumption, we appended a field to the LLDP packet indicating when it was transferred from the queue to the network card for transmission. After decoding this timestamp, the receiver can determine the one-way delay associated with this packet. More formally, let $T_{\text{send},i,s(i,r)}$ be the timestamp when LLDP packet i, destined for receiver node r, was transmitted by sender node $s(i,r)$, and let $T_{\text{receive},i,r}$ be the timestamp when that packet was received by node r. Let M_r denote the total number of packets received by node r, and assume there are N nodes in the network. Then, the average one-way delay is computed as:

$$\text{AvgDelay} = \frac{1}{\sum_{r=1}^{N} M_r} \sum_{r=1}^{N} \sum_{i=1}^{M_r} \left(T_{\text{receive},i,r} - T_{\text{send},i,s(i,r)} \right) \tag{5.1}$$

Regarding the network overhead introduced from control messages transmitted by cluster nodes, we measured it by monitoring the wireless card's interface in the operating system with the *tcpdump* tool and appropriately filtering the network packets to isolate traffic stemming from the Raft consensus algorithm as indicated by the used TCP port, and LLDP traffic as indicated by its Ethertype value within Ethernet

frames. We can formulate the overhead O_i in bps for each node i as:

$$O_i = \frac{\sum_{j=1}^{M} \left(S_{\text{Raft},j} + S_{\text{LLDP},j} \right)}{T} \times 8, \tag{5.2}$$

where M is the number of control messages (Raft and LLDP) captured during the measurement period T, $S_{\text{Raft},j}$ is the size of the j-th Raft message in bytes, and $S_{\text{LLDP},j}$ is the size of the j-th LLDP message in bytes. The total network overhead across the cluster is then the sum of the overhead across all nodes N:

$$O_{\text{total}} = \sum_{i=1}^{N} O_i \tag{5.3}$$

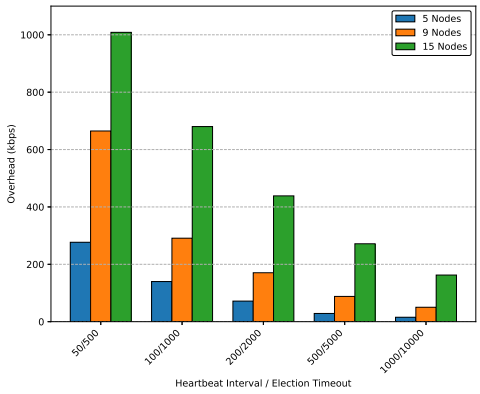
As regards the availability of the Master Controller, we consider it to be the most effective approach for assessing the robustness and adaptability of the SDN network. This is due to the fact that in the absence of a Master Controller for a given node, it is impossible to establish and forward new flows, which consequently leads to packet losses and/or significant delays. In order to determine the Master Controller's availability in each node, time was divided into one-second slots, and any slot in which a node failed to receive a heartbeat from a leader node of the Raft cluster (or the Master Controller from an SDN standpoint) was deemed *unavailable*. Then, for each node, the proportion of *available* slots (i.e. slots in which an MC exists and is reachable through the IP network) relative to the total number of slots is calculated and averaged. More formally, let T be the total observation time in seconds (i.e., the total number of one-second slots), A_i be the number of *available* slots for node i during the observation period and assuming there are N nodes in the network, then the Master Controller Availability (MCA) can be expressed as:

$$\text{MCA} = \frac{1}{N} \sum_{i=1}^{N} \frac{A_i}{T} \tag{5.4}$$

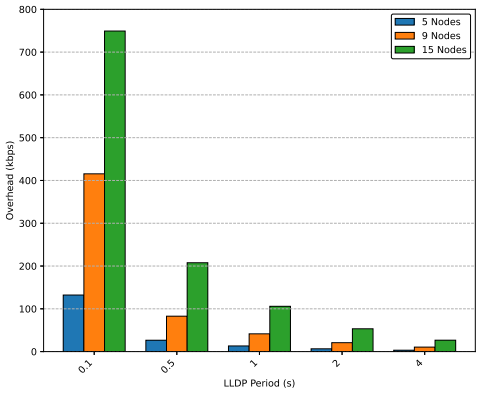
5.4.2 Experimental Analysis

The experiments were conducted by employing the Extendable Mobile Ad-hoc Network Emulator (EMANE) [73] and the Common Open Research Emulator (CORE), both of which provide a realistic environment for open source emulation. In order to generate mobility movement scenarios, Bonnmotion was utilized [74]. Table 5.1 contains details regarding the versions of software that were installed, the parameters of the emulation environment, and the specifications of the high-end computer that was utilized to conduct the experiments. We altered one single parameter in each series of experiments while preserving the default values for the remaining parameters, which are detailed in Table 5.2. In relation to the `heartbeat_interval` and `election_timeout`, we adhered to the recommended approach by setting the `election_timeout` to a duration that is ten times the `heartbeat_interval`, thus we altered both parameters simultaneously.

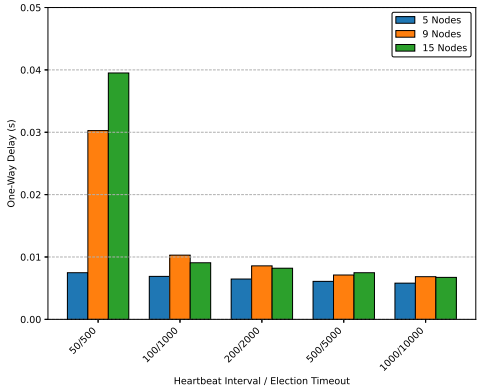
We measured the total network overhead and the average one-way delay for the `heartbeat_interval` and `election_timeout` parameters (that are jointly adjusted), as well as for the `lldp_period` under static conditions. This allowed us to primarily



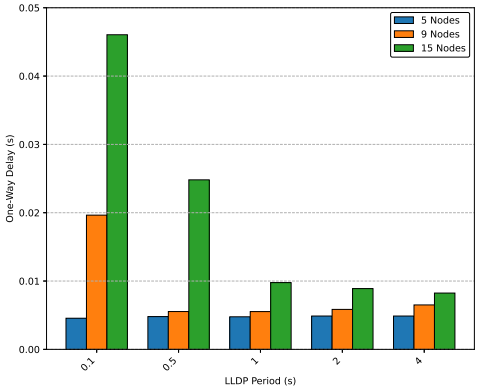
(A) Total Overhead for Raft



(B) Total Overhead for LLDP



(C) One-way Delay for Raft



(D) One-way Delay for LLDP

FIGURE 5.2: Plots for total overhead and associated one-way delay for both Raft algorithm and LLDP protocol for varying network sizes and for average node velocity of 0 m/s.

TABLE 5.1: Configurations used in the experiments

Experiment Parameters	Value
Processor	13th Gen Intel Core i9-13900KF @ 5.80GHz
Memory	64GB
Operating System	Ubuntu 20.04 server (Linux Kernel v5.4.0)
CORE Version	v7.5.2
EMANE Version	v1.2.5
Bonnmotion Version	v2.0
Ryu Version	v4.34
Etcd Version	v3.6.0
Radio Model	IEEE 802.11abg Model
PHY Rate	11 Mbps

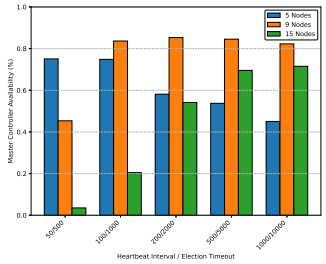
TABLE 5.2: Default values of parameters

Experiment Parameters	Value
`heartbeat_interval`	100 ms
`election_timeout`	1000 ms
`lldp_period`	2 s
`timeout_period`	10 s

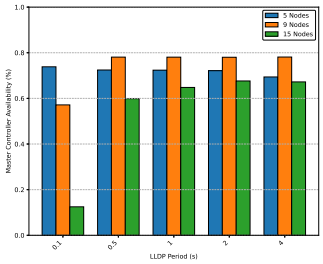
attribute any variation in one-way delay to the network overhead. The measurements were replicated for clusters consisting of five, nine, and fifteen nodes. Twenty experiments were conducted for every configured parameter value and cluster size, with each experiment lasting for a duration of 600 seconds. Regarding MCA, we followed a similar approach, however this time we exclusively examined mobility scenarios characterized by two different mean velocities: 5 m/s and 10 m/s.

The outcomes of the experimental analysis mentioned earlier are illustrated in Figure 5.2. In particular, Figure 5.2a illustrates how augmenting the time between heartbeats from 50 ms to 2000 ms leads to a substantial reduction in network overhead. This is also highlighted by the steep decline in the one-way delay as shown in Figure 5.2c, which drops to 81.6% for a fifteen-node cluster. It is noteworthy to mention that the considerable delay values observed in the case with a low `heartbeat_interval` can be primarily attributed to the long queue waiting times. These wait times arise from the increased volume of heartbeat packets and the low PHY rate (11 Mbps) utilized. We have opted for such a low rate to more accurately simulate practical tactical situations, wherein expanding the MANET's coverage area requires low modulations, and to emphasize the trade-offs inherent in low-capacity networks. An increase in the quantity of heartbeats not only results in queueing delays but also raises the probability of collisions. These collisions can be attributed to two factors: the simultaneous expiration of backoff timers or the hidden terminal phenomenon that is observed in CSMA/CA [1],[2]. Consequently, the re-transmissions that occur in the MAC layer are further affecting delays. A similar trend is observed when altering the `lldp_period` (see Figures 5.2b and 5.2d).

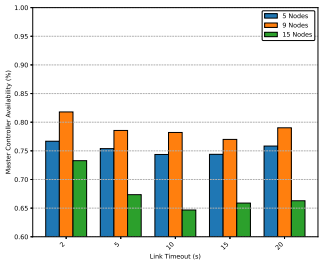
In relation to the MCA metric, it is apparent from Figures 5.3a, 5.3d, and particularly in the case of the small cluster, that reduced `heartbeat_interval` values and, consequently, `election_timeout` values result in improved availability. This is due to the network's prompt response to changes, which enables it to select a Master Controller that is more appropriate. In contrast, in the case of large clusters, where collisions



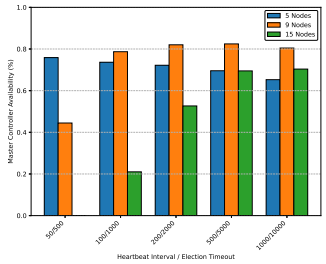
(A) Master Controller Availability for varying Raft parameters at 5 m/s.



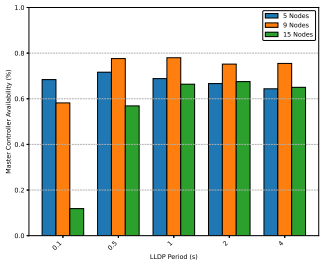
(B) Master Controller Availability for varying LLDP periods at 5 m/s.



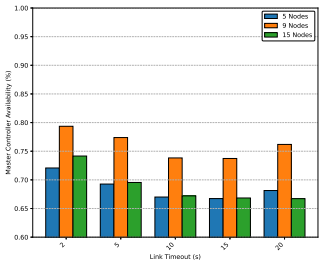
(C) Master Controller Availability for varying link timeouts at 5 m/s.



(D) Master Controller Availability for varying Raft parameters at 10 m/s.



(E) Master Controller Availability for varying LLDP periods at 10 m/s.



(F) Master Controller Availability for varying link timeouts at 10 m/s.

FIGURE 5.3: Plots for Master Controller Availability for varying Raft and LLDP-related parameter values, for varying network sizes and for varying average velocities of nodes.

and buffer overflows cause an unreliable network that struggles to select a Master Controller, increased `heartbeat_interval` values enhance performance until they become so high that the network becomes slow in its response to changes. When the `lldp_period` is modified, a same pattern is once more observed in Figures 5.3b and 5.3e. Furthermore, with respect to `timeout_period`, it is evident from Figures 5.3c and 5.3f that reduced values enhance MCA. This is due to the prompt detection of invalid links, which enables forwarding through alternative links and/or the selection of a more appropriate Master Controller. Conversely, larger values also exhibit improvement, given that links seldom expire despite being invalid. Therefore, in the case of temporary link invalidity, flow re-forwarding and re-elections of Master are avoided, and the previous Master's accessibility is restored once the links regain their validity. A final remark is that a cluster consisting of nine nodes demonstrates superior performance in terms of MCA when compared to clusters consisting of five and fifteen nodes. This is because the cluster of nine nodes provides numerous alternatives for nodes to establish connections with a Master Controller. Additionally, it avoids the substantial overhead of control messages that is associated with the larger cluster of fifteen nodes.

In conclusion, the experiments conducted reveal the critical influence of system parameter adjustments on both network overhead and overall performance in MANET environments. Thus, the importance of precisely balancing these parameters to guarantee the reliability and efficiency of the SDN network, particularly in dynamic and resource-constrained MANETs, is evident.

5.5 Self-Adaptive SDN Configuration

The analysis mentioned above offered an in-depth overview of the complexity linked to the proper configuration of the parameters that govern the operation of the designed SDN framework. Due to the innate tradeoffs and ever-changing nature of the MANET network, which, when combined with the 802.11 CSMA protocol and its accompanying phenomena like hidden terminals, create an extremely volatile environment, accurate modeling of the system becomes an extremely challenging endeavor. On the other hand, the cross-layer SDN approach and the large amounts of network data that can be obtained from the nodes through the OpenFlow protocol [50], make the model-free RL approaches very appealing for developing an intelligent, automated, and self-adaptive method for configuring the SDN network based on experience. In light of this, we have integrated RL agents into our system that dynamically configure control plane parameters based on observation of the network state across its layers.

5.5.1 Overall Architecture

Figure 5.4 provides an overview of the integration of RL agents into our architecture. As it can be observed from the figure, the agents reside on every node; however, only one, the one located on the MC's node, is active and involved in the decision-making. The rest are in a sleep state, ready to be activated whenever the node where they reside takes over the leadership. Besides the RL agent, there are also two other modules that are deployed on each node:

- **Network Monitoring Module**: This module is implemented as an SDN application over the Ryu framework. SDN *applications* are software modules that run on top of the controller for providing specific network functions. As stated in Section 5.3, every node runs a local SDN controller; therefore, the Network Monitoring Module runs on top of every LC. As its name implies, this application, is tasked with polling and collecting network performance-related statistics and metrics from both the OvS switch and the corresponding network layer, as well as from the driver of the wireless card and the corresponding MAC/PHY layer. These statistics include bytes and packets transmitted at both the network and MAC layers, packet delivery ratio (PDR), airtime utilization, and SINR values. Furthermore, as nodes are equipped with a GPS module, the module also collects the coordinates of the node's location and can therefore calculate its velocity. After being polled, the statistics are transmitted to the RL agent located at the MC. This exchange is OpenFlow compatible, as we are employing the standard-defined OpenFlow Experimenter messages that allow us to pass any arbitrary data. Specifically, the LC encapsulates the statistics within an Experimenter message, assigns it an identifier signifying that it relates to network statistics, and sends it to the OvS switch located at the node, which is where the MC resides. Due to the in-band design of the architecture, the OvS switch, after receiving the message, will forward it to the MC, which in turn will pass it to the RL agent's pre-processing module. This module collects statistics from all network nodes and generates the feature space for the RL agent.

- **Action Enforcement Module**: This module is also implemented as an SDN application and serves two purposes: a) transferring the RL agent's decision to the network nodes, and b) based on this decision, configuring the corresponding SDN framework and Raft implementation on each node. More specifically, the actions that are chosen by the RL agent are forwarded to the Action Enforcement Module of the MC. Subsequently, these actions are encapsulated within an OpenFlow Experimenter message that carries an identifier signifying that it is an action message. The message is then transmitted to every OvS switch within the network. Upon receipt of the Experimenter message, the OvS switch proceeds to forward it to the LC, which subsequently forwards it to the Action Enforcement Module. The module then performs subsequent API calls to the Topology Discovery mechanism and the Raft implementation (etcd) APIs, which we extended in order to provide the capability of on-the-fly configuration of `lldp_period`, `timeout_period`, `heartbeat_interval` and `election_timeout`.

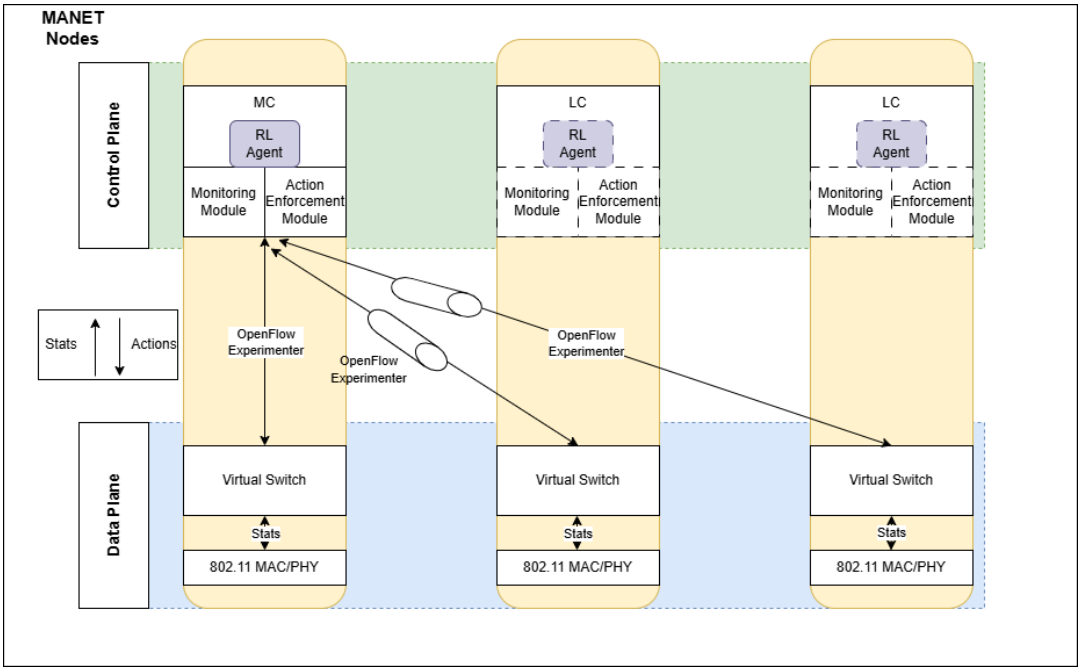


FIGURE 5.4: Integration of RL agent into the SDN-MANET framework.

5.5.2 Reinforcement Learning Framework

Reinforcement Learning is an artificial intelligence area that focuses on the development of algorithms that learn optimal behaviors via a structured framework of penalties and rewards. In contrast to supervised and unsupervised learning, which utilize structured datasets, RL utilizes an agent, a decision-maker, which iteratively refines its strategies via ongoing interaction with a dynamic environment.

As illustrated in Figure 5.5, throughout this iterative learning process, the agent is provided with critical information: the current state of the environment and a corresponding reward. Equipped with this information and its accumulated knowledge, the agent executes actions in a manner that optimizes its cumulative reward over an

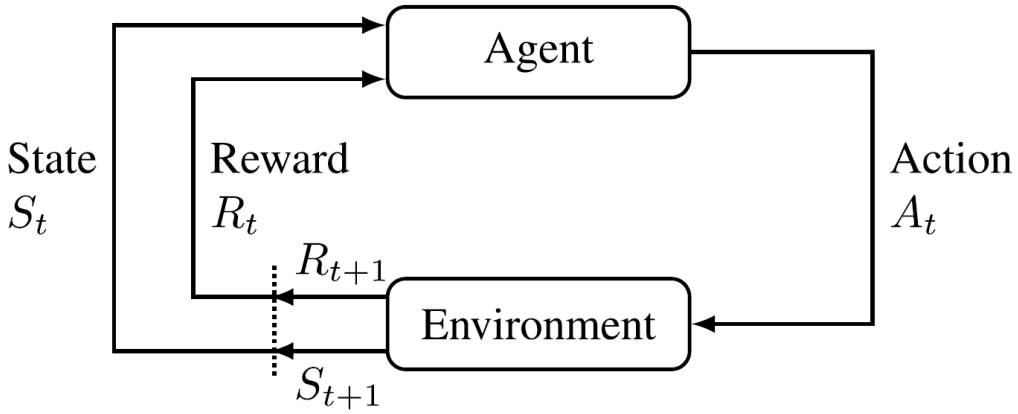


FIGURE 5.5: Interactions between agent and the environment [75].

extended period of time by navigating a sequence of state transitions. This interaction is not unidirectional; the agent's actions influence the subsequent state of the environment and the rewards it will receive, creating a complex feedback loop.

In contrast to supervised learning, which is directed by a clear external supervisor, the objective of RL is for the agent to autonomously determine which actions result in the greatest reward. This requires a delicate balancing between exploring new actions in order to determine their potential rewards and relying on established actions that are known for their substantial rewards. The agent's objective is to enhance its understanding and performance in the environment, striving for the highest possible long-term revenue by learning which actions, when taken in particular states, will lead to the most beneficial outcomes over time.

The mathematical formulation of the decision-making problem that the RL agent tries to solve is represented as a Markov Decision Process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}$ is the state transition probability matrix, $\mathcal{P}(s_{t+1}|s_t, a_t)$, denotes the probability of transitioning from state s_t to state s_{t+1} by taking action a_t, R is the reward function, $R(s_t, a_t)$, denotes the reward received after transitioning from state s_t by taking action a_t, and γ is the discount factor, $\gamma \in [0, 1]$. A policy, π, is a strategy that the agent employs to determine the next action based on the current state. $\pi(a_t|s_t)$ denotes the probability of taking action a_t in state s_t under policy π. The goal is to learn a policy π that maximizes the expected return, which is defined as the sum of discounted rewards obtained by following π:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))|s_0 = s\right] \tag{5.5}$$

The essence of Reinforcement Learning lies in its ability to adapt and make decisions in dynamic, and often uncertain, environments. By continuously learning from the consequences of its actions, the RL agent progressively refines its policy, leading to improved decision-making strategies and the ability to achieve its objectives more effectively.

5.5.3 Q-Learning

Q-Learning, an off-policy RL algorithm, seeks to ascertain the value of taking a given action in a particular state, denoted as the Q-value. It is considered as an off-policy RL algorithm because the Q-Learning function learns from actions that are outside the current policy, like taking random actions, and it does not need a model of the environment.

The core of Q-Learning is a Q-table or Q-function, $Q(s, a)$, which provides the quality (usefulness) of taking action a in state s to achieve a goal. Initially, Q-values are often initialized to arbitrary values (often zeros), and then updated iteratively through the learning process.

The update of the Q-values occurs with each step taken in the environment and is based on the Bellman equation:

$$\begin{aligned} Q(s_t, a_t) \leftarrow\ & Q(s_t, a_t) \\ & + \alpha \big[r_{t+1} \\ & + \gamma \max_a Q(s_{t+1}, a) \\ & - Q(s_t, a_t) \big] \end{aligned} \tag{5.6}$$

Here, s_t and s_{t+1} are the current and next state, respectively, a_t is the action taken at time t, r_{t+1} is the reward received after taking action a_t at state s_t, α is the learning rate, and γ is the discount factor. The policy π can be derived from the Q-function. At any state s, the action a to choose can be derived by selecting the action with the highest Q-value in that state. Furthermore, in Q-Learning, it is crucial to balance exploration (selecting new actions) and exploitation (choosing the best-known action). Often an ϵ-greedy strategy is employed, where with probability ϵ a random action a_t is chosen (exploration), and with probability 1 - ϵ, the best-known action, $a_t = \arg\max_a Q(s_t, a)$, is chosen (exploitation).

5.5.4 Offline Reinforcement Learning

Offline Reinforcement Learning, also known as batch Reinforcement Learning, focuses on training a policy from a large, fixed dataset of experiences without additional interaction with the environment, in contrast to traditional online Reinforcement Learning [76, 77, 78, 79]. This approach is particularly valuable when interacting with the environment is costly, risky, impractical, or the quantity of data that can be collected is significantly lower and less diversified. The dataset typically consists of tuples containing states, actions, rewards, and subsequent states, often collected from previous interactions of an agent (or multiple agents) with the environment, and can be denoted as $\mathcal{D} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^{N}$.

While offline RL presents the potential to leverage extremely large datasets gathered over a long period of time, it also presents a number of technical challenges. These challenges stem from two sources: the distributional shift, which may result in suboptimal policy performance in regions of the state-action space that were not previously observed in the dataset, and extrapolation errors that arise from the value function overestimating the value of state-action pairs that are not adequately represented. Consequently, these issues contribute to suboptimal policy decisions.

A recent approach that attempts to tackle these challenges is the Conservative Q-Learning, CQL [80], which estimates the value function $V^{\pi}(s)$ of a target policy $\pi(s)$ given a static dataset $\mathcal{D}$ that is generated by a behavior policy $\pi_{\beta}(a|s)$. CQL modifies the standard Q-Learning objective to prevent overestimation of Q-values by adding a regularization term alongside the standard Bellman objective. The objective function of CQL is given by the equation:

$$Q_{k+1} = \min_{Q} \left(\alpha \cdot \mathbb{E}_{s \sim D} \left[\log \sum_{a} \exp(Q(s,a)) - \mathbb{E}_{a \sim \pi_{\beta}(a|s)}[Q(s,a)] \right] + \frac{1}{2} L \right), \tag{5.7}$$

where Q_k represents the Q-values at iteration k, L is the standard Bellman objective function, and $\mu(a|s)$ is the desired distribution of action-states after training. Constant α is a hyperparameter of the optimization problem. The authors in [80] prove that for $\mu = \pi$, the resulting estimation of the value function and the Q-values satisfies the restriction $\hat{V}^{\pi}(s) \leq V^{\pi}(s)$ for all $s \in \mathcal{D}$, ensuring that every estimated Q-value is bounded by the actual value of the Value function, thus eliminating overestimation.

In simple environments with discrete and low-dimensional state and action spaces, CQL can be implemented with tabular methods where Q-values are stored in tables. However, tabular methods become impractical as the state and actions spaces grow, as they require enormous amounts of memory for the storage of the tables. In such complex environments, function approximators such as deep neural networks are employed to estimate the Q-values. Consequently, in our implementation we consider CQL for a discrete action space, specifically adapted for use with Double Deep Q-Networks (DoubleDQN) [81], which is a variant of the standard DQN algorithm that reduces overestimation of Q-values by using two separate networks to decouple the selection and evaluation of the bootstrap action in the Q-value update. The loss function that is used to calculate the gradients for the neural network and update the parameters of the network using the chosen optimizer is provided by the following equation:

$$\begin{aligned} L(\theta) = \alpha \mathbb{E}_{s \sim \mathcal{D}} \left[\log \sum_{a} \exp Q_{\theta}(s_t, a) \right] \\ - \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi_{\beta}(a|s)}[Q_{\theta}(s,a)] \\ + L_{\text{DoubleDQN}}(\theta) \end{aligned} \tag{5.8}$$

Here, θ are the parameters of the Q-function approximator (neural network). The loss function is composed of three terms:

- The first term is a regularizer that penalizes the Q-values by taking the log sum of exponentials across all actions. It pushes the Q-values to be lower unless supported by the data, thus mitigating overestimation.
- The second term is the expectation of the Q-values under the behavior policy π_{β} that generated the data. This term anchors the Q-values to those experienced in the dataset.

- The third term, $L_{\text{DoubleDQN}}(\theta)$, represents the standard loss function used in DoubleDQN, which typically includes a mean squared error between the Q-values predicted by the current network and the target Q-values computed from the next state's Q-values using a target network.

By combining these terms into a single loss function, CQL aims to train a policy that performs well on the given offline dataset while avoiding the pitfalls of overestimation, which can lead to suboptimal policies that overfit to the dataset and perform poorly in the actual environment.

5.6 Implementation

In this section we provide the implementation details of the actual RL agent that we trained and employed for dynamically configuring the parameters of our SDN framework. We first present the state space, which represents the environment that is given as input to the agent, and the action space that involves all the allowed configuration actions that the agent can execute. Then we design the reward that is the second input to the agent for evaluating the impact of actions towards our performance objectives and, finally, we detail our approach for the offline training of the agent.

5.6.1 State and Action Space

The state space $\mathcal{S} \subseteq \mathbb{R}^z$ corresponds to the features that are generated by the metrics polled from the Network Monitoring Module on all the network nodes. These metrics are then aggregated and pre-processed at the MC, thus summarizing the network information at time interval Δ_t. The features, in every time interval Δ_t constitute the state $s = [f_1, f_2, \ldots, f_{13}]$ of this interval and a subset of them (where applicable) is further represented by the tuple $\{f_{x,avg}, f_{x,std}\}$, where $x \in \{0, 1, \ldots, 13\}$, that contains the average and the standard deviation values respectively. In the following, we describe each of them, providing the notation for the average value and omitting the one for the standard deviation to save space:

- $f_1 = N$, where N is the size of the cluster of nodes in the SDN MANET network.
- $f_2 = V_{\text{avg}}$, represents the average velocity of the nodes.
- $f_3 = \frac{1}{N}\sum_{i=1}^{N} \text{avg}(\text{dist}(n_i, n_j))$, represents the average value of the average distance of each node from every other node in the network.
- $f_4 = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{j=1}^{F} T_{ij}\right)$, represents the average value of the outgoing throughput from every node and for every flow, where F represents the total number of flows, T_{ij} the outgoing throughput for flow j at node i and can be expressed as the total number of bytes transmitted for flow j divided by the time interval Δ_t, $T(j, \Delta_t) = \frac{\text{bytes}_{tx,j,\Delta_t}}{\Delta_t}$.

- $f_5 = \frac{1}{N}\sum_{i=1}^{N} T_{MAC,ij}$, represents the average value of the outgoing throughput from every node at the MAC layer, accounting also for possible retransmissions.
- $f_6 = \frac{1}{N}\sum_{i=1}^{N} \frac{\text{airtime}_i}{\Delta_t}$, represents the average value out of all network nodes of the airtime utilization perceived, meaning the fraction of time for which the channel was sensed as busy.
- $f_7 = \frac{1}{N}\sum_{i=1}^{N} \left(\frac{1}{P_{rx,i}}\sum_{p=1}^{P_{rx,i}} \text{SINR}_{ip}\right)$, represents the average value out of all network nodes of the average SINR received, where $P_{rx,i}$ is the total number of packets received by node i in the time interval Δ_t, and SINR_{ip} is the SINR of packet p received by node i.
- $f_8 = \frac{1}{N}\sum_{i=1}^{N} \left(\frac{P_{ack,i}}{P_{tx,i}}\right)$, represents the average value across all network nodes of the Packet Delivery Ratio at MAC layer, where $P_{ack,i}$ is the number of successful and $P_{tx,i}$ the number of total packet transmissions of node i.
- $f_9 = \frac{1}{N}\sum_{i=1}^{N} \left(\frac{P_{SINR,i}}{P_{rx,i}} \times 100\right)$, represents the average percentage of packets dropped due to low SINR value.
- $f_{10} = \frac{1}{N}\sum_{i=1}^{N} \left(\frac{P_{RXTX,i}}{P_{rx,i}} \times 100\right)$, represents the average percentage of packets dropped due to being received simultaneously with a node's transmission.
- $f_{11} = \frac{1}{N}\sum_{i=1}^{N} \left(\frac{P_{queue,i}}{P_{rx,i}} \times 100\right)$, represents the average percentage of packets dropped due to queue overflow.
- $f_{12} = \frac{1}{N}\sum_{i=1}^{N} \left(\frac{P_{dupl,i}}{P_{rx,i}} \times 100\right)$, represents the average percentage of packets dropped due to being duplicates of another reception.
- $f_{13} = \frac{1}{N}\sum_{i=1}^{N} \left(\frac{P_{hidden,i}}{P_{rx,i}} \times 100\right)$, represents the average percentage of packets dropped due to a collision with a hidden terminal's transmission.

The action space is $\mathcal{A} = \{a_1, a_2, ..., a_{125}\}$, where each action a_i corresponds to the configuration of specific values for all the considered parameters. Specifically, we consider a configuration tuple for both {`heartbeat_interval`, `election_timeout`} as they are tightly related. The configuration values for each parameter are provided in Table 5.3. Consequently, each action represents a unique combination of the three parameters and given that there are five different configuration values for each, we encode it as a base-5 number. In total, the size of the action space is 125. We have kept the action space limited in order to enable the efficient training of the agent given the capacity of our computing resources and the limited time to generate the datasets.

TABLE 5.3: Configuration values of parameters

Parameter	Values
`heartbeat_interval`/ `election_timeout`	[0.05/0.5 s, 0.1/1 s, 0.2/2 s, 0.5/5 s, 1/10 s]
`lldp_period`	[0.1 s, 0.5 s, 1 s, 2 s, 4 s]
`timeout_period`	[2 s, 5 s, 10 s, 15 s, 20 s]

5.6.2 Reward Function

The reward function is the means for evaluating the impact of agent's actions. As such, we have designed a reward that ensures that the agent's objectives align with the desired network performance outcomes. Specifically, the reward function encourages the agent to minimize the control messages overhead between the nodes and, simultaneously, maximize the availability of MC, while also maintain consistent performance across the network by minimizing the standard deviation of both metrics. It can be expressed as:

$$R = w_a \times MCA - w_v \times SD(MCA) - w_o \times NO - w_s \times SD(NO) \tag{5.9}$$

where:

- R is the reward given to the agent at each step.
- MCA is the average Master Controller Availability, defined in Equation (5.4).
- $SD(NA)$ is the standard deviation of the Master Controller Availability, reflecting the variability in the availability of a controller.
- NO is the normalized average control messages overhead between the nodes, which is defined in Equation (5.2).
- $SD(NO)$ is the standard deviation of the normalized control messages overhead.
- w_a, w_v, w_o, and w_s are weights that reflect the relative importance of average availability, availability variability, average overhead, and overhead variability, respectively.

By controlling the values of the weight variables, one can favor availability over control overhead and vice versa, considering the requirements of the applications running over the MANET.

5.6.3 Training

As mentioned in Section 5.5, we opted for the offline training of our RL agent, and specifically the CQL algorithm, as we considered it more practical due to our ability to generate various scenarios that include a plethora of network topologies and mobility patterns, thus creating a very diversified dataset that will allow the agent to generalize better. This can lead to a more robust policy that handles both common and rare network states effectively, without requiring the agent to wait for these conditions to occur naturally in an online setting. Furthermore, learning from a large dataset in a batch mode leads the agent to faster convergence since it can repeatedly iterate and refine its policy. Then, once deployed, the agent operates with an already-learned optimal or near-optimal policy, bypassing the lengthy learning phase associated with online RL. Finally, offline training is employed for another practical reason: Although RL agents are deployed on each node of the network, only one of them, the one located on the MC's node, is active at any given time. As a result, the agents would be unable to train at the same rate in the case of online learning, resulting in parameter configuration decisions that are highly unbalanced.

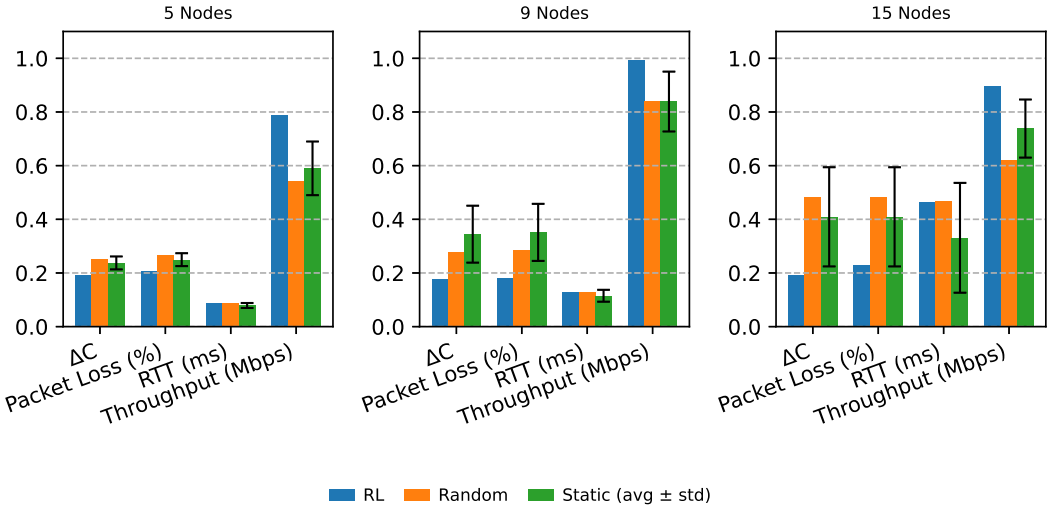


FIGURE 5.6: Normalized Performance with Static Strategy Variance (15 Configurations).

The dataset generation was facilitated by the employment of CORE and EMANE that allowed the accurate emulation of the MANET environment as well as the ability to monitor and capture network statistics from across the network layers. Furthermore, Bonnmotion was employed for creating the mobility patterns that the emulated network nodes were following. In order to limit biases and the aforementioned challenges from the distributional shift, we followed a random exploration policy, while generating the dataset. Although this approach is not less sample efficient compared to a behavior policy, it is useful for learning a generalized policy, so that the agent can handle the various situations that can emerge due to the volatility and dynamic environment of MANETs. In particular, we generated 800 scenarios, each one of them with duration of two hours and with a step of one minute. Each scenario was considered as an episode. The scenarios differentiated from each other in terms of number of nodes, network traffic load, average velocity of the nodes and mobility models (Random Waypoint vs Random Walk). The final dataset was preprocessed in order to clean up potential invalid values stemming from parsing errors or EMANE bugs, as well as for normalizing features and rewards, and was fed to the Discrete CQL algorithm for the training of the agent.

5.7 Evaluation

In this section, we provide a comprehensive evaluation of our RL-based SDN autoconfiguration framework and highlight its benefits when compared against random and static parameter configurations. The evaluation was performed through rigorous emulations using CORE and EMANE, which provided realistic MANET environments, with each experiment scenario lasting for one hour and repeated at least ten times in order to ensure robustness and statistical reliability.

5.7.1 Experimental Setup

The experiments involved dynamic MANET scenarios, emulating various mobility speeds, network traffic loads and network sizes. Specifically, we created three different scenarios, each corresponding to a network size of 5, 9, and 15 nodes, respectively. The mobility pattern was generated using Bonnmotion and involved various average node speeds that alternated every 10 minutes over the 60-minute duration of the entire experiment. Similarly, every 5 minutes, a random number of links between node pairs were activated in order to exchange TCP traffic using *iperf*. For each scenario, we evaluated our RL-based parameter configuration strategy, which configures node parameters every minute based on the environment, alongside a random strategy and a static configuration strategy. For a fair comparison, a fixed seed was used for random generators. In the static configuration strategy, the parameters assessed were `heartbeat_interval/election_timeout`, `lldp_period` , and `timeout_period` with the configuration values as shown in Table 5.3. The default values for `heartbeat_interval/election_timeout` were 500/5000 ms, for `lldp_period` was 2 s, and for `timeout_period` was 10 s. In total, for each scenario, we evaluated 15 static configuration strategies.

5.7.2 Performance Metrics

In order to better assess the performance of the network when employing the considered strategies, we required a metric that could isolate the impact on performance of the configuration strategies from the effects of uncontrollable events, such as nodes moving out of range. We needed to ensure that SDN-related performance variations were not simply the result of changing physical connectivity. Since our SDN framework is responsible for establishing links and routes at the IP layer, we want to evaluate how effectively the SDN adapts to changes at the PHY layer. In other words, we want to measure the gap between perceived connectivity at the IP layer and the actual connectivity at the PHY layer. To this end, we define the following key metric, **Connectivity Deviation**, which forms the basis of our evaluation:

$$\Delta C \;=\; \frac{\sum_{i \neq j} \left| \; L_{i,j} \;-\; [P^*]_{i,j} \right|}{n\,(n-1)}\,,$$

where

- $L_{i,j} \in \{0,1\}$ is the logical adjacency (SDN-established reachability) between nodes i and j,
- $[P^*]_{i,j} \in \{0,1\}$ is the entry in the transitive-closure of the physical adjacency matrix (true PHY-layer reachability), and
- n is the total number of nodes.

This metric quantifies the difference between the logical connectivity at the SDN (IP) layer and the underlying physical connectivity between the nodes. Logical connectivity is determined by building an adjacency matrix based on SDN-established forwarding paths, while physical connectivity is derived from the adjacency matrix generated using physical layer data exported from EMANE via the use of *emanesh*

tool. By computing the *transitive closure* of the physical adjacency matrix P, denoted P^*, we obtain a reachability matrix in which

$$[P^*]_{i,j} = 1 \iff \exists \text{ a (possibly multi-hop) path in the PHY-layer graph connecting nodes i and } j.$$

This ensures that ΔC accounts for all indirect physical links, not just direct neighbors.

To construct the logical adjacency matrix, we periodically employ the *ping* utility to exchange ICMP packets between the nodes. Specifically, we transmit four ICMP packets from each node to every other node in the network, assuming bi-directionality, and consider a link established if at least two transmissions are successful. With the comparison of these adjacency matrices over time, we are able to distinguish between SDN-induced effects and those caused purely by changing physical network conditions. By construction, ΔC measures the fraction of node pairs whose SDN-level connectivity disagrees with the underlying physical connectivity:

- A high ΔC indicates that SDN has *failed* to establish logical paths across existing physical links.
- A low (or zero) ΔC shows that SDN has successfully discovered and exploited all available physical links.

Overall, Connectivity Deviation serves as an important diagnostic metric that allows us to assess the added value-or potential cost-of the SDN control plane in MANETs.

While Connectivity Deviation captures effectively the adaptability to changing physical network conditions of the SDN framework and of its configuration strategies, it is used in conjunction with traditional network performance metrics to ensure that the impact of network overhead introduced by the different configuration strategies is also taken into consideration. These additional metrics, measured at the SDN (IP) layer, include Packet Loss, average Round-Trip Time (RTT) and Throughput.

5.7.3 Results and Discussion

Figure 5.6 presents the min–max normalized performance of our RL-based auto-configuration framework ("RL"), a random parameter strategy ("Random"), and the static strategy ("Static") across three network sizes (5, 9, and 15 nodes). All four metrics—Connectivity Deviation (ΔC), Packet Loss, RTT, and Throughput—are normalized to $[0, 1]$ for direct comparison on a single plot. In the static case, each bar shows the mean over 15 fixed-parameter combinations, and the error bars indicate one standard deviation, highlighting the variability introduced by any single static setting.

For the **5-node** network, RL reduces Connectivity Deviation by roughly 24% compared to random and by 19% compared to the static mean. Packet Loss falls by about 22% versus random and 17% versus static. Round-Trip delay under RL matches the

random baseline (within 1%) and lies within the static spread, while Throughput increases by 46% over random and by 34% over static.

When **scaling to 9 nodes**, RL's gains become even more pronounced: Connectivity Deviation drops by 37% relative to random and 49% relative to static. Packet Loss follows a similar pattern and is reduced by 36% compared to random and 48% compared to static. Delay improves by about 1% versus random and remains as low as the best static setting (but without its variability), and Throughput is 18% higher than both the random and static baselines.

In the most challenging **15-node** scenario, RL cuts Connectivity Deviation by 60% versus random and 53% versus static. Packet Loss decreases by 53% compared to random and 44% compared to static. Although absolute delay in large topologies rises, RL still trims RTT by 1.4% versus random (while remaining within the static range). Throughput under RL surpasses random by 44% and static by 21%, demonstrating robust capacity maintenance under high mobility.

Overall, the static strategy's large error bars underscore the lack of robustness in any fixed parameter set. Moreover, the particular static parameter set that yields the best performance at 5 nodes is different from the optimal set at 9 or 15 nodes, underscoring that no single fixed configuration generalizes well across varying network scales. In contrast, our RL-based framework yields a single, repeatable configuration that consistently outperforms both random and static baselines across all network sizes.

5.8 Conclusion

In this chapter, we presented a novel Reinforcement Learning (RL)–based framework for the autonomous self-configuration of key control-plane parameters in Software-Defined Networking (SDN) deployed over Mobile Ad Hoc Networks (MANETs). Building on our prior cross-layer, in-band SDN architecture, the RL agent continuously tunes heartbeat intervals, LLDP discovery rates, and Raft election timeouts in response to real-time link-quality, throughput, and mobility statistics.

Through extensive CORE/EMANE emulations in one-hour mobility scenarios with 5, 9, and 15 nodes, we showed that the RL approach consistently outperforms both random parameter selection and 15 exhaustive static configuration sets. Specifically, RL achieves up to a 60 % reduction in connectivity deviation, cuts packet loss by over 50 % and boosts throughput by 20–46 %, all while exhibiting negligible run-to-run variance. These gains grow more pronounced as network size and dynamism increase, demonstrating RL's ability to balance the inherent trade-offs between responsiveness and control-plane overhead under highly volatile conditions.

Our results confirm that model-free RL is a practical and effective technique for making SDN viable in resource-constrained, high-mobility MANET environments. Further extensions could involve the application of decentralized, multi-agent RL, where each node runs its own learner and coordinates via federated updates—to improve scalability and resilience in large or partitioned MANETs. Another extension could involve the incorporation of energy-consumption metrics into the reward function, for trading off performance gains against device lifetime for battery-powered nodes.

Chapter 6

On the Implementation of a Software-Defined Memory Control Plane for Disaggregated Datacenters

Contents

6.1 Introduction 72
6.2 Related Work 73
6.3 Overall Hardware Architecture 74
6.4 Software-Defined Memory Control Plane 75
6.5 Graph Model Design 77
6.6 Evaluation 83
6.7 Conclusion 85

By adopting a disaggregated hardware architecture, datacenters can achieve considerable efficiency gains and transition to a more sustainable and green future. By decoupling resources from a single monolithic server and connecting them through a high-speed optical network, it is possible to significantly increase resource utilization and reduce power consumption by consolidating workloads into fewer resource units. In this chapter, we design and develop a software-defined control plane for disaggregated memory datacenters. Its core component, the Software-Defined Memory Controller, is the orchestrating software, which efficiently materializes the disaggregation concept. It accomplishes this by managing and monitoring remote resource pools, allocating resources to workloads, instrumenting the dynamic configuration of the underlying optical network for interconnecting remote compute and memory resources and interacting with software agents residing in host and guest Operating Systems for coordinating the attachment of remote memory. A major contribution of our design is the minimization of the delay for scheduling workloads and Virtual Machines in a disaggregated datacenter, which is accomplished with the efficient modelling of the disaggregated resources and networking elements into a graph for retrieving configuration data, as well as the optimization of the graph implementation. The architecture is based on the Everything-as-a-Service paradigm and is tightly coupled with OpenStack, the leading cloud infrastructure management software. Evaluation experiments validated the employment of a graph database system by the Software-Defined Memory Controller by demonstrating 58 percent faster query times than relational databases for a small-to-medium-sized datacenter, with the percentage increasing as the size of the datacenter grows.

6.1 Introduction

Modern cloud datacenters continue to suffer from resource underutilization, resulting in increased costs of ownership and significant energy waste. Recently published real-time traces of Google's datacenters [82] demonstrate that, while average resource utilization has increased since 2011, primarily due to increased consumption of workloads, the issue persists. And it will continue to exist as long as traditional datacenter architectures' inherent limitations remain unresolved. As a significant constraint, the mainboard tray and its hardware components continue to serve as the fundamental building block upon which operating systems, middleware, and applications are built. The distribution of datacenter resources (CPU cores, memory, and network capacity) is determined during design and is largely static, as it is rarely updated, except for hardware upgrades. On the other hand, resource demands for workloads exhibit a dynamic and highly fluctuating distribution, with studies indicating workloads with memory-to-processor ratios exceeding four orders of magnitude [82]. Modern systems heavily rely on resource virtualization and application containerization to overcome this barrier and improve utilization, energy efficiency, and total cost of ownership, while also enabling workload migration and isolation. However, the ability of Virtual Machines (VMs) and containers to meet resource demands is still constrained by the boundaries defined by a single motherboard tray and the resources that it hosts, precluding workload scaling and elasticity.

The aforementioned limitations can be addressed by implementing resource disaggregation in cloud datacenters. By decoupling resources from monolithic server trays and connecting them via a low-latency Software-Defined network, resource disaggregation enables greater granularity in resource allocation. Thus, it empowers the dynamic creation of a platform composed of components located in distinct pools to meet the resource requirements of individual workloads. Such an architecture has the potential to significantly increase utilization while also providing the ability to allocate the maximum possible amount of a datacenter resource (CPU, memory). Additionally, it provides significant energy savings by allowing unused resources to be powered down individually.

In [83, 84] authors present the project dReDBox in which they implement a modular low-power architecture that enables memory disaggregation by delivering the hardware platform and complementing software layers that enable low-latency transparent remote memory access and a full-fledged Type-1 hypervisor over the disaggregated memory.

This chapter provides the design and implementation of a centralized Software-Defined Memory (SDM) Control plane for such an architecture. The SDM Control plane acts as an orchestrator, coordinating all the underlying operations necessary for realizing disaggregated memory access and making it directly available to datacenter administrators and end users in a completely transparent manner via integration with common resource management and network operating systems such as OpenStack. It enables the following capabilities in particular:

- Fine-grained management and monitoring of disaggregated resources.
- Configuration, routing, and establishment of optical network interconnections between compute resources and remote memory pools dynamically and on-demand.

- Allocation and attachment of remote memory on running VMs and physical hosts.

In addition to implementing a framework for managing disaggregated resources, our most significant contribution is the development of a graph model that enables the rapid and scalable retrieval of configuration data. Furthermore, we implemented and validated this model in both a relational and a graph database and designed an algorithm that considerably reduces the time required for data retrieval, hence reducing the time required for workload scheduling.

The remainder of the chapter is organized as follows: Section 6.2 overviews prior relevant work on disaggregated datacenter architectures and the related challenges. Section 6.3 presents the overall hardware architecture of a disaggregated system such as [85]. Section 6.4 discusses the SDM Control plane's implementation details, while Section 6.5 describes our graph model and the optimization of its implementation. In section 6.6, an evaluation of key performance factors of the SDM Control plane is detailed. Section 6.7, concludes the chapter by summarizing our proposed scheme and findings and outlining future directions for work.

6.2 Related Work

Disaggregated datacenter architectures have been intensively studied in recent years to identify their benefits, restrictions, and technical challenges. The disaggregated architecture has many benefits, including modularity, higher resource utilization, [86], [87], and energy-efficiency due to reduced power consumption [88]. Further studies indicate that memory underutilization in High Performance Computing (HPC) systems [89] can benefit from memory disaggregation [90]. However, despite its benefits, disaggregation is associated with a non-trivial performance penalty when accessing remote datacenter resources, on top of an additional cost for deploying the optical interconnect.

A key enabler for effective resource disaggregation architectures is the network, which must be designed in such a way that is flexible, dynamic and re-configurable and it can support acceptable application performance while also allowing for low-latency communication between remote resources [91], [92]. In overall, hardware disaggregation architectures adhere to the software-defined principles, which allow for the on demand (re)-configuration of hardware in order to meet resource requirements, moving from the traditional rigid and inflexible way that traditional datacenters are currently built [93], [94].

Besides defining the architecture, the hardware design and the network interconnection that are required for realizing disaggregation, several works investigate the problem of efficient and optimal resource allocation to VMs in order for infrastructure and service providers to reap the associated benefits of the increased resource utilization. In the paper [95], authors develop a reinforcement learning agent that meets quality of service requirements while also providing energy savings, whereas in [96], three workload-aware policies are presented for scheduling and placement of workloads in disaggregated systems that significantly reduce the number of missed deadlines.

In this chapter, we focus on the implementation details of a fully-fledged, software-defined disaggregated hardware controller and orchestrator, responsible for managing and monitoring resources and for dynamically configuring network interconnections. Additionally, our work gives significant consideration to the effective modeling of the disaggregated hardware and network components in order to achieve the necessary scalability for managing resources in large datacenters as well as to the required optimizations for offering low and pragmatic times for the completion of scheduling decisions. This is in contrast to the aforementioned similar works, which only focus either on the definition of the high-level disaggregated architecture or on the design of efficient algorithms for workload placement and scheduling.

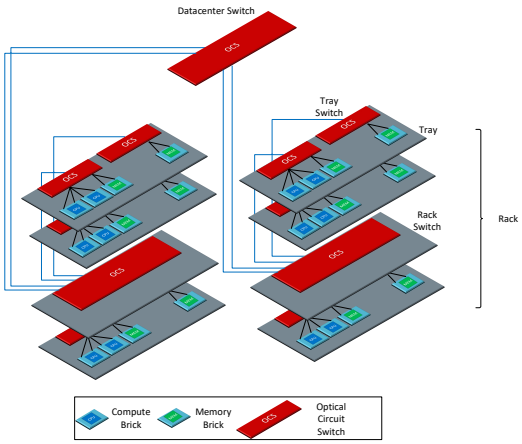


FIGURE 6.1: Hardware Architecture

6.3 Overall Hardware Architecture

The physical separation of datacenter resources and the formation of interconnected pools that allow the allocation of arbitrary combinations between them form the foundation of the disaggregation concept. To that end, the proposed architecture [85] that is depicted in Figure 6.1 decomposes the resources into basic, self-contained components of a single type of resource referred as *bricks*.

The *Compute brick* is the system's fundamental processing unit, including a multi-processor system on-chip (MPSoC). Furthermore, it contains a small amount of low-latency off-chip memory required for booting an Operating System (OS), several serial transceivers for interconnection with remote bricks, and an Ethernet module for connecting to the control and management network.

The *Memory brick*, on the other hand, is the fundamental unit for providing memory resources, which can then be distributed among *Compute bricks* and, finally, to the OS and VMs hosted on them. It is comprised of DDR memory modules, which form the byte-addressable local address space, as well as a number of serial transceivers.

The bricks are attached to *trays* that have a standard rack mount system size and can hold a total of 16 bricks in any random combination. A *rack* is composed of *trays*, and several *racks* constitute a complete disaggregated datacenter. Interconnection between bricks is accomplished through the use of low-latency high-speed optical networks based on software-controlled circuit switches on *tray*, *rack* and *datacenter*

levels, as a three-tier hierarchical Clos network, with their ports connected to brick transceivers and between them, providing intra-tray, inter-tray and inter-rack connectivity.

In the context of this disaggregated architecture, a memory segment is a memory portion of a *Memory brick* that may be allocated to any operating system or application running on a *Compute brick*. Disaggregated memory segments, once allocated, are provided to the *Compute brick* as byte addressable chunks of main memory, directly mapped into its physical address space and transparently accessible through standard memory bus transactions. Further details can be found in [84].

6.4 Software-Defined Memory Control Plane

In contrast to traditional datacenters and their state-of-the-art resource management software, the disaggregated datacenter introduces two new challenges: i) distributing the globally accessible memory address space across the *Compute bricks*, and ii) interconnecting the bricks via controllable optical switches. Both tasks are needed to create a logical server from remote disaggregated resources that will host and execute user workloads. To meet these requirements, we designed and developed the Software-Defined Memory Control plane, which monitors remote memory availability, configures switches to establish circuits between bricks, manages resources and orchestrates remote memory attachment to host and guest OS, and schedules workloads performance-aware and power-efficiently. The SDM Control plane consists of the SDM Controller, which implements the previously mentioned functionalities, the SDM Agent, which runs on the *Compute bricks* and interacts with the SDM Controller and host OS to attach/detach remote memory, and the *Platform Synthesizer*, which applies SDM Controller configuration commands to the underlying network. Our approach interfaces directly with OpenStack, which is utilized as an Infrastructure-as-a-Service provider in cloud datacenters.

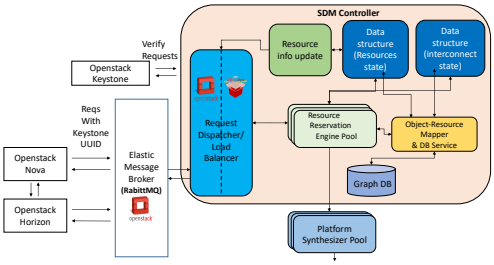


FIGURE 6.2: SDM Controller

The SDM Controller is the disaggregation orchestrator's heart. It's written in Python and comprises of autonomous, containerized services, following to Everything-as-a-Service concepts (XaaS). The interfaces are REST APIs, allowing integration with various services and software modules. Northbound interfaces interact with *OpenStack Nova*, whereas Southbound interfaces communicate with *Platform Synthesizer*, which applies configuration commands to hardware and optical switches. The SDM Controller is the core component that supports resource reservation, monitoring, and dynamic multi-rack reconfiguration.

Figure 6.2 depicts the SDM Controller's blueprint. It interfaces with the OpenStack Nova components via the AMQP message broker, RabbitMQ, following the publish/subscribe paradigm. Authorization and authentication of message exchanges is performed by the OpenStack Keystone service. Users issue requests for spawning/deleting VMs through OpenStack Horizon, the web dashboard, and in order to satisfy a potentially increased load, containerization allows us to dynamically start/stop multiple instances of the SDM Controller services. By employing a *Request Dispatcher/HTTP Load Balancer*, such as NGINX, the requests are routed to the appropriate instance of *Resource Reservation Engine* in order to maintain performance. The *Resource Reservation Engine's* roles are as follows: i) receiving resource requirements from OpenStack Nova, ii) inspecting availability and making a power-consumption conscious selection of resources, iii) safely reserving the selected resources while avoiding race conditions, and iv) generating all required configurations that will then be delivered to *Platform Synthesizer*. In parallel, the *Resource Info Updater* is responsible for updating utilization metrics and remote memory attachment information for each *Compute brick*, which is then displayed in the corresponding tab of OpenStack Horizon.

The SDM Controller employs two distinct data structures (shown in Figure 6.2) that are used by the *Resource Reservation Engine*. The first models disaggregated resources and is used for performing the "logistics" part of resource reservation and allocation, while the second models the interconnection of hardware components, allowing for fast retrieval of configuration parameters for optical network programming. Both structures are further described in the following paragraphs:

- ***Resource Allocation State:*** In practice, this structure uses in-memory hash tables saved to disk for persistence. It keeps sorted lists of all available *Compute* and *Memory* bricks objects. Sorting ascends by available CPU cores in *Compute bricks* and RAM memory in *Memory bricks*. Also, both *Compute* and *Memory* brick variants have two lists of memory segment objects. The first list is free segments, and the second is reserved. Objects are ordered sequentially on both lists to allow the memory driver to allocate/deallocate continuous memory. The segments are shifted using incremental memory reservation or releasing. Each segment object has a base address that occupies a unique segment size section in each brick's local memory layout and a unique identifier that distinguishes it throughout the entire disaggregated system. Additionally, CPU cores are kept in lists. In essence, resource allocation structures store memory and CPU core utilization. With the proposed sorting method, components can be quickly identified and reserved, optimizing system usage.

- ***System Interconnect State:*** The second data structure maintains the state of the system's interconnections: which *Memory bricks* are connected to which *Compute bricks*, via which transceivers and switch ports. To manage this type of information, we consider that a graph database is the most appropriate choice. Not only because the number of possible paths aggressively scales with the number of transceivers and switching layers, but also because the graph structure can provide the best possible performance during queries for available connectivity paths between disaggregated components with some simple optimizations. In fact, we employ a hybrid combination of a graph database, implemented with JanusGraph [97] and a custom in-memory path cache, which is a hash-table containing all reserved paths. This improves overall performance by reusing existing active paths to support new allocations.

Furthermore, the *Object-Resource Mapper and DB Service* is responsible for mapping the representations of the resources as used by the aforementioned data structures to the data elements stored in the database, as well as translating the requests to the query language used by the database and subsequently executing the queries and returning the results to the *Resource Reservation Engine*.

To put it all together, the SDM Controller's *Resource Reservation Engine* first selects a *Compute brick* from the utilization-sorted list and then selects *Memory bricks* sorted by locality (intra-tray, intra-rack, inter-rack) and utilization ratio. When all workload resource requirements are met, it inspects the graph interconnect cache for available paths. If there is not, a graph query retrieves available paths. If this fails for all or a subset of the selected *Memory bricks*, it continues with another *Memory brick*, preserving any memory selections from the previous *Compute brick*. When no *Memory brick* is reachable, the Engine discards the current *Compute brick* and tries the next most used one. The algorithm we implemented reduces fragmentation by filling resource gaps on heavily used bricks before moving to the next free brick. It also prioritizes intra-tray bricks to reduce latency. This has both positives and negatives. SDM Controller requires a lot of CPU time and has less throughput than OpenStack resource schedulers (that of course do not take memory disaggregation into account). Its response time to a reservation query depends on the size of the infrastructure, the system's current reservation state, and the number of graph queries required for a successful allocation. However, this conservative approach allows for facilitating more workload (while maintaining performance), especially when memory or CPU pressure is high (i.e. overall rack deployment utilization is more than 80% percent). Since the objective of disaggregation is to optimize resource utilization, we opted for the described algorithm and optimized its execution.

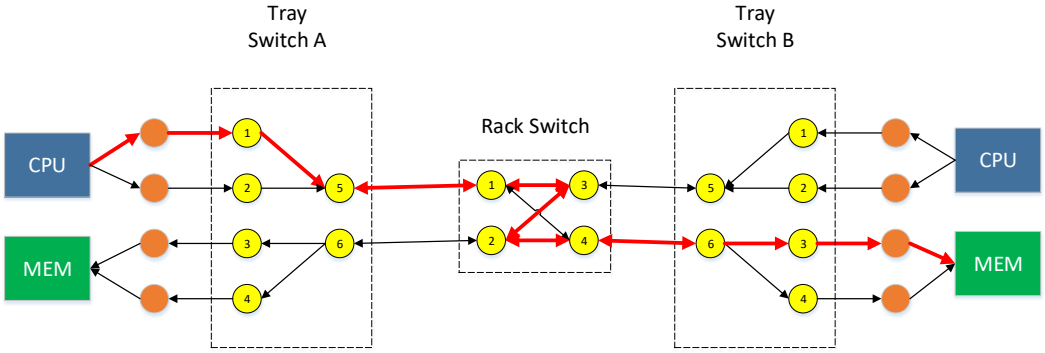
6.5 Graph Model Design

This section provides an overview of the graph model design that is employed in order to hold the interconnect state data as well as its implementation details and optimizations using widely-adopted database systems.

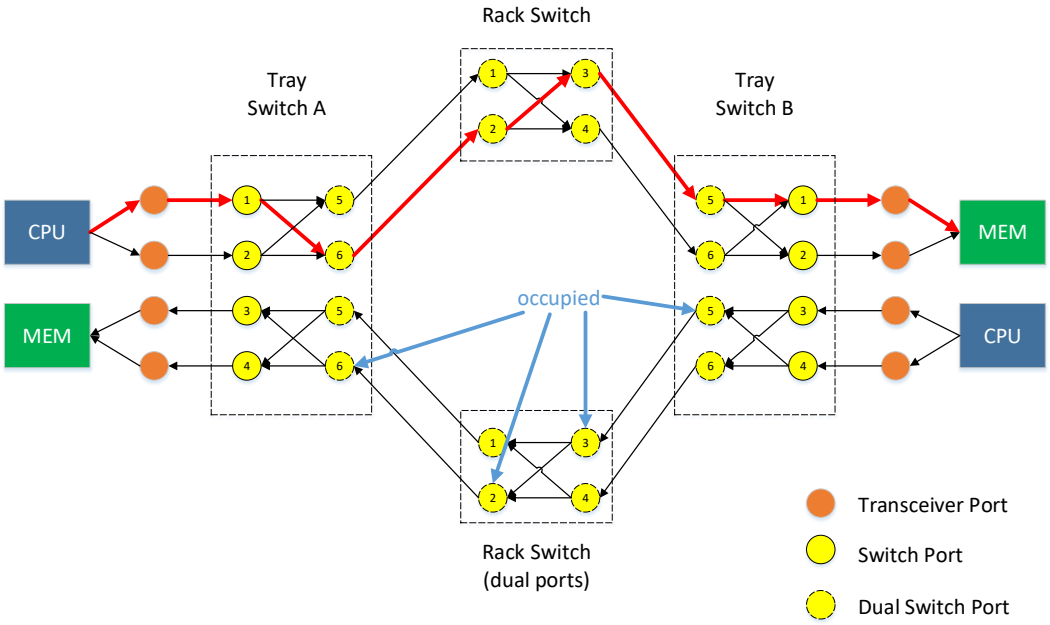
6.5.1 Data Schema

The interconnect state graph data schema is illustrated in Figure 6.3 and is described as follows: *Compute bricks*, *Memory bricks*, transceivers and switch ports are modelled as vertices and the edges are used to represent a direct connection between them. A path is formed by neighboring vertices and their accompanying edges (red line). It denotes the presence of a potential (multi-hop) connection between remote resources. We employed and implemented the following concepts to create a graph that can be efficiently queried and return a path containing configuration parameters for the establishment of inter-connectivity:

- We opted for an acyclic directed graph design.
- We completely eliminated the possibility of loops and inspection of non-meaningful paths, simply by controlling edge placement during the construction phase.



(A) Without dual ports



(B) With dual ports

FIGURE 6.3: Graph representation

To implement a directed acyclic graph, all *Compute bricks* were regarded as roots and all *Memory bricks* were regarded as sinks. On every possible graph path, directed edges can only point outwards from *Compute bricks* and inwards to *Memory bricks*. With a single switching layer, this is straightforward as each port's *brick* type information is readily available. However, this information is lost when two or more switching layers are interleaved between *Compute* and *Memory* bricks. The up-link of the prior switching layer can carry either *Compute brick* or *Memory brick* traffic, so both directions must be supported. To model multi-switching layers, we developed the notion of ***dual ports***, as shown in Figure 6.3. This figure shows a simple rack configuration with two trays and two switching layers (2 tray switches and 1 rack switch) between them. Each *tray* contains *Compute bricks* and *Memory bricks*, making the construction of an acyclic directed graph challenging. Without dual ports (Figure 6.3a), inspecting potential paths between bricks may contain loops that slow

down query execution. The dual port concept is materialized during the deployment phase graph generation by duplicating intermediate switching layer ports to provide independent directed paths in both directions. This approach requires also replicating only the up-link ports of the first-tier switches (tray switches), which are inside the tray and directly connected to *bricks*. As shown in Figure 6.3b, each tray switch's up-link ports are replicated, while the rack switch is completely replicated. This way, multiple switching layers provide separate, legitimate paths between each *Compute* and *Memory brick*. Physical switch ports are actually single instances, and the dual ports representation is used to provide opposite directions at the graph level. Due to this fact, dual port pairs that refer to the same physical port are associated in the graph database schema so that when one is reserved, the other cannot be used with other path allocations. In the example shown in Figure 6.3b the blue lines indicate the dual ports that are considered reserved after the reservation of the path indicated by the red line.

Finally, our method generates graph edges carefully to reflect all valid paths. For instance, each switching layer has input and output ports that correspond to a single physical port's two roles. An input port can never also be an output port, so we don't build edges from input ports to output counterparts. According to our modeling, *Compute bricks* are always connected to input ports and *Memory bricks* to output ports. When a port role is always input, it won't have an output counterpart. All of these enhancements are considered during graph construction, as they will result in improved run-time performance.

6.5.2 Database System Implementation

In this subsection, we describe our database system implementation that optimizes query execution times and, subsequently, scheduling decision times. We also justify our choice of a NoSQL database, JanusGraph, a distributed graph database, over a relational SQL database, PostgreSQL, for storing data for the management and configuration of the disaggregated system. In order to compare the two approaches, we first modeled the directed acyclic graph in a relational database. As in [98], we define two tables: one for nodes (bricks, transceivers, and switch ports) and another for edges.

- nodes {nodeID, attributes}
- edges {edgeID, node_src, node_dst}

For the evaluation of databases' performance, we deployed our SDM Controller implementation alongside JanusGraph 0.5.2 and PostgreSQL 11.12 on a server featuring an Intel Xeon processor at 3.7 GHz frequency with 12 cores and 64 GB of RAM. Initially, we generated three tiny versions of a *datacenter* configuration on the SDM Controller data model. Each tiny version is comprised of two *racks*, each of which contains two *trays* equipped with a *Compute brick* and a *Memory brick*. Each brick features eight transceivers which are directly connected to the first layer of switching that is located inside the tray. In our evaluation, we considered a realistic selection for the optical switches, which corresponds to a 48-port switch for the *tray* level and a 384-port switch for both *rack* and *datacenter* levels. For each configuration version, the three switching layers (*tray, rack, datacenter*) are configured with an oversubscription ratio of 9:3:1, 4:2:1 and 1:1:1, respectively.

TABLE 6.1: Basic query execution time.

Database	9:3:1			4:2:1			1:1:1		
	Tray	Rack	DC	Tray	Rack	DC	Tray	Rack	DC
PostgreSQL	6.78s	6.94s	6.65s	48.87s	47.02s	44.87s	∞	∞	∞
JanusGraph	2.06s	17.3s	60.91s	2.08s	44.45s	286.97s	2.38s	∞	∞

TABLE 6.2: Optimized query execution time.

Database	9:3:1			4:2:1			1:1:1		
	Tray	Rack	DC	Tray	Rack	DC	Tray	Rack	DC
PostgreSQL	0.71s	1.36s	1.96s	0.79s	1.37s	2s	0.86s	1.48s	2.14s
JanusGraph	3.17s	3.35s	3.21s	3.44s	3.55s	3.54s	3.67s	3.86s	3.72s

Basic query

The first series of experiments examines the performance of a simple, non-optimized query for retrieving the shortest path between a *Compute* and a *Memory* brick in JanusGraph and PostgreSQL, using Gremlin Query Language of Apache TinkerPop [99] and PL/pgSQL [100], respectively. For the PL/pgSQL code, we used a recursive Common Table Expression (CTE) that builds a table with all paths through each recursion until it reaches the destination node (*Memory Brick*). In detail, the non-recursive term finds all nodes directly connected to the starting node by an edge. Then, first repetition of the recursive term finds nodes directly connected to the starting node's one-hop neighbours. However, nodes reached by a shorter path during a previous repetition are not added to the growing ultimate result set. When the recursion reaches the destination node, it returns all the paths that led there. Then, the result set is trimmed and only the shortest paths are kept. Using Gremlin, with a lot simpler query, we similarly traverse edges from the source node (*Compute Brick*) to the destination node.

Table 6.1 shows query execution times for both databases. We considered three scenarios for retrieving the shortest path between bricks on the same *tray* (Tray), on different *trays* but on the same *rack* (Rack), and on different *racks* across the datacenter (DC).The infinity symbol denotes an execution in which the query did not yield any results due to a crash during retrieval, either due to disk space exhaustion in PostgreSQL or an Out Of Memory exception in JanusGraph. This naive approach to graph querying generates huge intermediate tables for the relational database and stretches the JanusGraph heap space for the graph database. It is evident that when the network is less oversubscribed and, subsequently, the graph grows, query execution time increases. We should also note that JanusGraph uses batch processing, meaning that queries are Breadth-First. This is why finding the path between bricks in the same tray is faster than in different trays or racks. On the other hand, PostgreSQL's recursive CTE prevents access to intermediate tables containing discovered paths. Thus, recursion cannot be terminated early when the target node is visited, so all paths must be found first.

Algorithm 1 Find Shortest Path with Pruning

```
1: procedure FINDSHORTESTPATH(startNode, endNode, E(G))
2:     n ← 0
3:     prevPaths ← (startNode, e.node_dst),
       ∀ e ∈ E(G), where e.node_src = startNode
4:     allPaths ← prevPaths
5:     while true do
6:         tempPaths ← {(p, e.node_dst) ∈ prevPaths × E(G) | p ∼ e}
           (= terminal(p)),
           where e.node_dst ∉ prevPaths
7:         n ← rowCount(tempPaths)
8:         if n < 1 OR endNode ∈ tempPaths then
9:             exit
10:        end if
11:        for terminal(p) ∈ tempPaths do
12:            dedupPaths(tempPaths, terminal(p))
13:            if terminal(p) ∈ allPaths then
14:                delete(tempPaths, p)
15:            end if
16:        end for
17:        prevPaths ← tempPaths
18:        allPaths ← allPaths ∪ tempPaths
19:    end while
20:    pathToDest(allPaths, endNode)
21: end procedure
```

Optimized query

The configuration retrieval time using a straightforward but naive approach for querying the graph shows that it is not possible to scale efficiently with configurations that correspond to realistic real-world deployments of several racks each consisting of several trays in a datacenter. This is due to the large number of supernodes in the graph, each of which represents a cross-connected switch port to all other ports of the switch. Considering that a rack or datacenter switch has 384 ports and the number of switches increases as the datacenter grows, there could be millions or billions of paths to inspect between two nodes in a real-world deployment, all of which must be stored in memory or persistent storage while the query is executing. As a result, such a query is inefficient.

To overcome this limitation, we developed an optimized algorithm to prune early and temporary results, saving time and space. Starting with PostgreSQL, we implemented a PL/pgSQL procedure similar to the previously described recursive CTE algorithm, but with access to intermediate results. Using this access we perform early path pruning that reduces the size of intermediate tables. Algorithm 1 describes the process of finding the shortest path between two nodes using a recursive CTE emulation with early path pruning. The table *allPaths* represents the growing result set in memory, while *prevPaths* and *tempPaths* store ephemeral results of each recursion. The procedure begins by emulating the non-recursive term that finds all

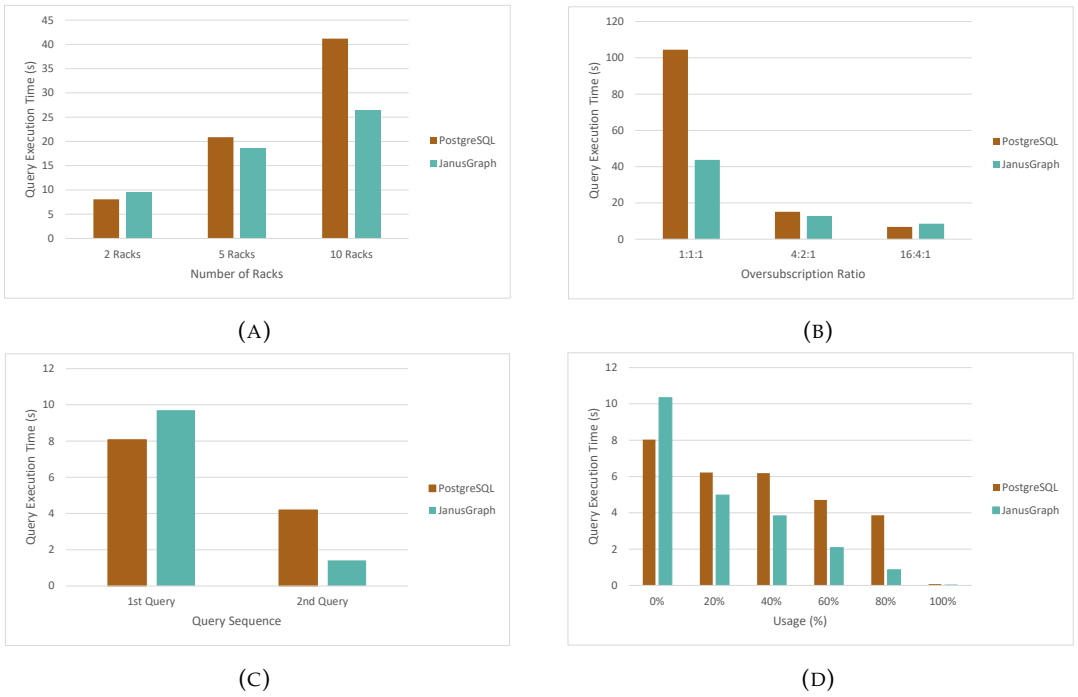


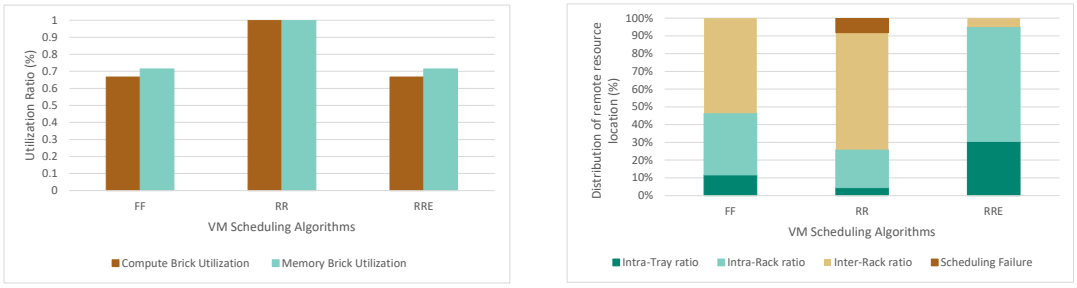
FIGURE 6.4: Query execution times for (a) varying number of racks, (b) varying oversubscription ratios, (c) effect of caching in sequential queries, (d) effect of utilization of switch ports

directly connected nodes to the starting node. The recursive term is emulated with a while loop, in which all paths discovered in earlier recursions are enlarged by adding nodes connected to the paths' *terminals* (the last visited nodes), if they have not already been visited. Emulated recursion ends when graph nodes are exhausted or the target node is found. In each loop iteration, we perform early path pruning, removing all but one path to each distinct new *terminal* from the *tempPaths* table and all newer and longer paths to previously discovered *terminals* from the *allPaths* table. The paths discovered during this recursion and stored in *tempPaths* are added to *allPaths* after pruning.

Gremlin provides a considerably more straightforward and human-readable syntax for the graph database in JanusGraph, and so the implementation of a similar query that supports early path pruning is much easier with the traversal shown below.

```
g.V('ComputeBrickID').store('x').
repeat(out().where(without('x')).dedup().
aggregate('x')).
until(hasId('MemoryBrickID')).limit(1).
path()
```

Using the optimized queries, we performed again the previous experiment and the results are given in Table 6.2. Early results indicate that the approach scales well as the graph grows. PostgreSQL appears to produce shorter query execution times in this tiny configuration, but this will be investigated in the next section. Additionally, access to PostgreSQL's intermediate results allows us to stop inspecting paths earlier when we reach the target node, implementing a Breadth-First-Search, while Gremlin's internal implementation mechanics render it a Depth-First-Search, although batch processing is still used.



(A) (B)

FIGURE 6.5: (a) Utilization of *Compute* and *Memory* bricks, (b) Distribution of remote resource location.

6.6 Evaluation

In this section we provide the evaluation of the SDM Controller implementation. In specific, we examine characteristics and design choices that affect performance in terms of both scheduling time and scheduling efficiency for VM provision as well as dynamic memory allocation.

6.6.1 Scheduling Time

Scheduling time refers to the time required for a) selecting the appropriate *Compute brick* for hosting the VM, b) selecting one or more *Memory bricks* with sufficient resources from which the SDM Controller allocates disaggregated memory, c) retrieving the configuration data for establishing the interconnect between the *bricks*, and d) applying the configuration on the *bricks*. It is evident that the retrieval and application of configuration data are the delay parameters that heavily influence total scheduling time. As the latter depends on memory hotplug implementation on the host OS as well as the switch configuration time, we focus on the former.

To compare the database systems under consideration, we ran a series of experiments to evaluate performance under larger datacenter configurations of the SDM Controller data model, as well as the effect of caching and a more complex query that inspects port usage. In this series, we considered a data model configuration with several full *racks*, each with 12 *trays* containing 8 *Compute Bricks* and 8 *Memory Bricks*. Figure 6.4 depicts the evaluation results. In the first experiment (Figure 6.4a), we generated three configurations of 2, 5, and 10 *racks*. All configurations had 9:3:1 oversubscription ratio. As we can observe, JanusGraph provides a substantial performance benefit in larger *datacenter* configurations and larger graphs, and is 36% faster than PostgreSQL in a *10-rack datacenter*. In the second experiment (Figure 6.4b), we compared 1:1:1, 4:2:1 and 16:4:1 oversubscription ratios in a *2-rack datacenter*. Again, we notice that JanusGraph significantly outperforms the relational database by 58% in a non-oversubscribed network, which requires more switches and a larger graph. In the third experiment, we investigated the effect of caching (Figure 6.4c). JanusGraph caches vertices and a subset of their adjacency list (properties and edges) in a database level cache, whereas PostgreSQL caches table data in 8KB pages. Both database caches were set to 2GB and we loaded a configuration of a *2-rack datacenter* with 9:3:1 oversubscription ratio. We executed queries twice and

observed caching's speedup. The results indicate that JanusGraph caching is significantly more efficient, with the second query being 85% faster than the first, whereas in PostgreSQL improvement is around 48%. Finally, we created a more complex query that searches for the shortest path between two nodes by inspecting only paths without already-used switch ports or transceivers. We modeled usage by adding a feature (vertex attribute for the graph database and column in the nodes' table for the relational database) that stores node occupancy. We loaded an identical configuration as in the last experiment, and Figure 6.4d shows that JanusGraph performs significantly better when querying for a shortest path across unoccupied ports and transceivers. Overall, the results demonstrate that employment of a graph database to store system interconnection state is appropriate and justifiable, as JanusGraph performs better in larger graphs, which correspond to real-world datacenter deployments with multiple racks. These findings agree with [101], where graph database showed better performance in a simple shortest path query, especially as the level of nodes between source and destination increased. In addition, our findings show that the SDM Controller's Resource Reservation Engine can leverage caching efficiency in subsequent allocations for VMs hosted in the same *Compute Brick*.

6.6.2 Scheduling Efficiency

To maximize the benefits of disaggregation, VM scheduling must maximize resource utilization to concentrate workloads in fewer *bricks* and reduce power consumption by powering off the idle. On the other hand, scheduling must consider performance penalties when remote resources are in different trays or, even worse, racks. The SDM Controller's Resource Reservation Engine (RRE) optimizes both goals by following the procedure described in Section 6.4. Its effectiveness is compared to a First Fit (FF) approach, which allocates workloads to the first available *bricks* that can meet demands, and OpenStack's default Round-Robin (RR) scheduler, which spreads VMs evenly across all hosts.

For performing the comparison between the three aforementioned approaches we loaded a configuration of *4-rack datacenter*, each with twelve *trays*, featuring eight *Compute bricks* and eight *Memory bricks*. Each *Compute brick* has 4 cores and each *Memory Brick* has 64 GB of RAM. The oversubscription ratio is set at 9:3:1. We then scheduled 460 VMs requesting a combination of one of the three flavors of CPU cores with one of RAM MBs shown in Table 6.3, with equal probability.

Figure 6.5a demonstrates that both FF and RRE algorithms consolidate workloads into the minimum required resources (66% *Compute bricks* and 71% *Memory bricks*), allowing idle resources to be powered off to conserve energy. Figure 6.5b illustrates how efficient scheduling decisions are to performance. We can observe the distibution of VM allocations within a *tray*, *rack*, or *datacenter*. The RRE algorithm that considers brick locality achieves 95% rack-scale allocations, while the other two fail to deliver adequate performance with datacenter-wide allocations. We also notice that high fragmentation prevents the RR algorithm from scheduling 8% of VMs. In overall, the SDM Controller's scheduling process is capable of striking a delicate balance between utilization and performance.

TABLE 6.3: Different flavors of CPU cores and RAM

Flavor	Cores	RAM (MB)
Small	1	16384
Medium	2	32768
Large	4	65536

6.7 Conclusion

Memory disaggregation in datacenters will offer to infrastructure and cloud service providers modularity, increased resource utilization, and improved energy efficiency. A critical enabler is a Software-Defined Control plane that enables efficient management, orchestration, and interconnection of disaggregated resources.

To that end, we designed and implemented the SDM Controller. Its main duties are: a) Given the resource requirements for a workload to be deployed, it selects the resources that will participate in the allocation by considering both the power consumption and the memory latency; b) it orchestrates all the necessary operations for dynamic configuration of the optical switches to establish a circuit interconnection between the selected resources as well as the operations for attaching remote memory to host and guest Operating Systems. Our design decisions, which aim to optimize scheduling time and scheduling efficiency were validated through an extensive evaluation. Results showed a significant reduction in scheduling time, an important factor that has not received any attention in the related literature to our knowledge.

Chapter 7

Multi-Cluster Orchestration of 5G Experimental Deployments in Kubernetes over High-Speed Fabric

Contents

7.1 Introduction 87
7.2 Background and Motivation 89
7.3 Implementation Design 91
7.4 Evaluation 94
7.5 Conclusion 97

Network Functions Virtualization has been a key enabler for the wide adoption of cloud-native functions for workloads. Well-established orchestration frameworks, such as Kubernetes, optimize network operation to meet the networking requirements of deployed workloads, while providing a flexible API for fine-grained control throughout the lifecycle of the workload. As a result, these tools can be used effectively to provide access to distributed 5G experimental facilities, even across continents. However, resource clusters may belong to distinct administrative authorities; therefore, cluster integration must occur by exposing only the necessary information and services. In this chapter, we suggest employing SUSE Rancher for managing multi-cluster Kubernetes deployments and utilize the Submariner framework in order to securely export services and establish connectivity across clusters. We evaluate the efficacy of such integrated framework and its various configurations over a high-speed networking fabric (up to 25 Gbps) connecting the various clusters and enabling 5G and beyond experimentation.

7.1 Introduction

Fifth-generation networks (5G) have completely transformed our digital world interactions and communications. They are the primary enablers for applications requiring higher data rates and ultra-low latency, including autonomous vehicles, augmented reality, and smart cities. 5G represents a paradigm shift from traditional wireless networking by softwarizing and virtualizing network functions (SDN/NFV) and multiplexing them with a cloud-native approach of containerized microservices to deliver fine-grained control and management. This approach enables

network operators to reduce operational costs, enhance system flexibility and scalability, and simultaneously enable the development and provisioning of services and applications with variable throughput and latency requirements. Utilizing softwarization and virtualization/containerization, the distributed deployment of a 5G network is now possible, where the cloud-native 5G Core Network can be instantiated in a cloud environment with abundant resources, while the 5G RAN can be deployed at the network's edge to enable real-time, low-latency applications.

Parallel research initiatives from standardization consortia, organizations, industry, and academia have begun to address the limitations of 5G and evolve it into the 6th generation (6G). To this end, research infrastructures and testbeds play an important role by providing experimental platforms that enable realistic evaluation of concepts, protocols, and services to researchers from academia and industry. In addition, they offer a controlled environment in which researchers can manage the deployment of their experiments and fine-tune their parameters in a standardized, well-defined, and softwarized manner, thereby ensuring reproducibility.

To facilitate truly large-scale, realistic experimentation, however, researchers require a variety of frequently highly heterogeneous high-capacity resources. The availability of such resources in research infrastructures exists on a global scale, but the problem is that these infrastructures are more isolated silos of resources than a unified research space. This results in a fragmented view of the total available resources and hinders the deployment of fully distributed large-scale applications.

To address this challenge, it is necessary to integrate the various research facilities into a single one that provides an environment for the seamless and simple access to heterogeneous experimental resources from various sites and locations around the globe. In such a situation, a central hub is required for the management and orchestration of resources and experiments, while direct access to individual sites for the same operations is also required.

For such an integration, virtualization and containerization of network functions (Virtual/Container Network Functions) and services will be essential to realizing its potential, because they will decouple the network from its physical infrastructure and heterogeneity. Additionally, the adoption of orchestration frameworks such as Kubernetes [102] is essential for assisting in bridging the gaps in integration between the various research sites.

In this thesis, we design and implement a fully-fledged multi-domain orchestration framework for providing centralized orchestration and fine-grained control of CNFs, VNFs and services over Kubernetes-based resource clusters, thereby facilitating the discovery and interconnection of services existing in distinct clusters. We employ an experimentally driven approach in order to determine the overhead and tradeoffs of the adopted solution, using isolated clusters in the NITOS testbed, the Greek node of the SLICES-RI platform [103]. We assess this setup by quantifying the performance impact of such a distributed deployment and discuss the different deployment options and limitations that they present.

The remaining sections of this chapter are structured as follows: The section 7.2 provides a summary of relevant concepts and background information. Section 7.3 describes in detail each of our contributions while presenting the overall architecture of our developed framework. In section 7.4, a proof-of-concept use case deployed on

multiple clusters is used to evaluate the framework. Section 7.5, concludes the chapter by reviewing our proposed scheme and findings and identifying future research directions.

7.2 Background and Motivation

In this section, we present some background concepts regarding virtualization and containerization, as well as their application in the cloud-native approach of the 5G Service-Based Architecture (SBA), where control plane functionality is provided by an interconnected chain of Network Functions (NFs).

7.2.1 Virtualization and Containerization

By enabling the creation of virtual instances of hardware platforms, storage, and network resources, virtualization enables the decoupling of physical resources from the underlying hardware. It enables the concurrent execution of multiple such instances, usually virtual machines (VMs), on commercially available hardware (server, server cluster), while providing isolation and security to software operating on them. In addition, it offers cloud providers flexibility and considerably improves the utilization of their hardware resources by enabling the optimal placement and migration of virtual machines, which contributes to meeting SLA requirements. Another advantage of virtualization is the increased availability of services, as hardware failures can be quickly recovered by migrating or replicating VMs to another physical machine. All of the aforementioned advantages have resulted in a significant reduction of CAPEX and OPEX for infrastructure providers, and have played a significant role in the development of cloud technologies that provide on-demand services and applications by sharing a pool of computing, storage, and network resources.

Containerization is a well-established technology that service providers and developers are swiftly adopting. Containerization is fundamentally a lightweight form of virtualization, wherein a container is a self-contained software package with its dependencies. The major difference between a container and a virtual machine lies at the level of abstraction. Containers are software abstractions of the operating system, whereas virtual machines are software abstractions of the hardware platform. The kernel space of the operating system is shared by containers running on the same host machine, despite the fact that their processes are entirely isolated and protected. Containerization is gaining popularity because it offers a significant advantage over virtual machines, namely that containers are more lightweight and have less overhead than VMs, which include a complete operating system stack and are consequently much slower to boot. This provides enhanced portability and scalability, efficient load balancing, rapid deployment, and overall improved resource utilization and efficiency. For all these benefits, the NFV industry and mobile operators are starting to favor container-based platforms and Containerized Network Functions (CNFs) over the conventional VM-based NFV platforms and the traditional Virtualized Network Functions (VNFs).

7.2.2 Network Disaggregation

At the same time, 5G networks have introduced a number of innovations in terms of network deployment and overall operator flexibility, enabled by the adoption of the concept of disaggregation. 5G introduces two forms of disaggregation by implementing: 1) the functional split of the base station units to simpler elements (e.g. CU/DU/RU split), with the CU/DU components being able to be executed at the cloud/edge; and 2) the Control/User-plane disagreggation across the different elements at the stack (e.g. CU disaggregation to CU-U and CU-P communicating over the E1 interface, or the O-RAN interfaces for programming the network using xApps). Through the adoption of Service-Based Architecture and the execution of discrete network functions, 5G Core Network components are also being disaggregated. As a result, cellular network operators can greatly benefit from the virtualization of their components, which enables their network to be executed in a cloud-native manner, thereby reducing their capital and operational costs and making the deployment of the network more flexible.

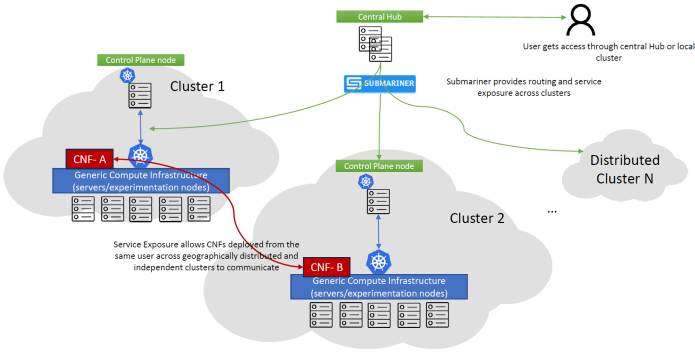


FIGURE 7.1: The overall under-study architecture: we consider geographically distributed clusters that operate independently. On top, the framework adds a management layer through a central hub, and exposes network functions and services to other clusters.

7.2.3 Workload Orchestration

The Management and Orchestration (MANO) of these virtualized functions is of utmost importance to the network operator, as it ensures the proper chaining of services, the establishment of appropriate datapaths among them, and their seamless operation during their lifecycle. There are several options for the orchestration of services, depending on their operation and the environment for which they are intended. The Kubernetes (K8s) ecosystem is one of the most prominent frameworks for orchestration. K8s automates the instantiation, scaling, and management of container-based applications and microservices, while specialized plugins facilitate the network establishment and service exposure within the cluster. K8s claims to be one of the most successful open-source orchestration platforms compatible with all modern container technologies at present. K8s provides integrated monitoring functionalities for monitoring and restoring application health, efficient resource allocation and storage organization, automated load-balancing and replication of high-traffic containers, management of application runtime state and control of deployments and updates, enabling the efficient use of physical resources [7, 6].

Although K8s provides a well-structured API for deploying workloads, it can only manage operations within a single cluster. There is no organized off-the-shelf solution for integrating other clusters (possibly under different administration domains) into a single centrally managed facility. In this work, we progress beyond these limitations and employ a fully-fledged solution that enables the cross-domain orchestration of several K8s clusters that are administered by distinct administration authorities. We develop the functionality for securely exposing services across integrated clusters and conduct experiments with workloads across the integrated resource continuum. In the following section, we describe the overall architecture and components that enable this functionality.

7.3 Implementation Design

In this section, we provide the design and implementation of our framework that facilitates the deployment and orchestration of the 5G Core Network and the 5G RAN across multiple distributed research infrastructures (multiple domains) by exposing CNFs and providing transparent inter-cluster connectivity.

7.3.1 Overall Architecture

Figure 7.1 is a visual representation of the overall architecture under consideration. It is comprised of several independent research infrastructures, displayed as nodes, each of which represents a resource island. A container management system based on Kubernetes manages each cluster's of resources directly. On top of the clusters a central hub exists, where a multi-domain orchestrator is deployed and directly interfaced with the individual Kubernetes-based clusters, enabling the deployment and management of containerized workloads across domains. The experimenters can login in from the central hub to a graphical user interface that allows them to inspect clusters and their worker nodes, deploy and configure pods and deployments, monitor the status of infrastructure and workloads, and define security and network policies. However, the configuration is flexible in the sense that the experimenters can potentially log in directly to the cluster management portal of an individual node with the same credentials and deploy their experimental workload locally.

Researchers can leverage such an architecture in order to experiment with existing and develop new concepts on 5G networks disaggregation and slicing in a large-scale multi-domain environment. Open-source implementations assist them, with the OpenAirInterface5G platform (OAI) [104] for the 5G RAN and Core components, srsRAN for the RAN [105], and Open5GS and free5GC [106] for the Core being the most well-known frameworks. Although all implementations support the majority of functionalities, OAI has a larger user base and provides additional features, such as disaggregation of RAN and support for multiple SDR devices.

7.3.2 Multi-domain Orchestrator

The European Telecommunications Standards Institute's (ETSI) NFV architectural framework forms the basis for NFV specification and management in 5G networks. Specifically, the ETSI NFV MANO specification [107] defines the framework under

which VNFs are provisioned, deployed, configured, and managed throughout their life-cycle. This framework is comprised of three major functional blocks: a) the NFV Orchestrator (NFVO), b) the VNF Manager (VNFM), and c) the Virtualized Infrastructure Manager (VIM). Initially, VNF management was primarily focused on virtual machine handling, but CNF support has since been introduced. Several compliant frameworks, such as OSM and ONAP [108], have been developed to implement the NFV MANO architecture, but a cloud-native approach is required to offer multi-domain orchestration support over existing containerized infrastructures and provide a flexible and simple method for managing multi-site experimentation.

To this end, we propose employing SUSE Rancher as the orchestrating framework for VNFs and CNFs. Rancher is a Kubernetes management tool that enables the deployment and management of clusters on any provider or location, as well as allowing the import of existing, distributed Kubernetes clusters that operate in different regions, cloud providers, or on-premises. It supports multiple flavors of Kubernetes, including K8s, K3s, and RKE/RKE2, and one of its main advantages is its centralized authentication and role-based access control (RBAC), which enables administrators to manage multiple clusters and their members from a central location. Though it is primarily targeting the deployment of CNFs, VNF deployment is also possible through extensions such as KubeVirt [109]. Deploying Rancher on a central hub to manage multiple Kubernetes-based clusters across distributed regions enables access to these clusters, management of their members and nodes, management of their persistent volumes and storage classes, management of their projects, namespaces, and workloads, and deployment and configuration of monitoring, logging, and alerting tools.

Figure 7.2 illustrates the high-level architecture of Rancher, in which the majority of Rancher's software components reside on the Rancher server. These components coordinate to provide access to multiple downstream clusters. Before initiating API calls to the master of the downstream Kubernetes cluster, a Rancher user must first authenticate with the Rancher server, and specifically with the Authentication Proxy, which adds the proper Kubernetes impersonation payload. The communication between Rancher and Kubernetes clusters is performed using a service account that identifies pod-running processes. A Cluster Agent is deployed on each downstream cluster to establish a tunnel with the dedicated Cluster Controller on the Rancher server. The Cluster Agent connects to the Kubernetes API of its cluster, manages workloads and pods, implements global policy roles and bindings, and communicates with the cluster controller regarding health status, events, and statistics. The Cluster Controller is responsible for configuring access control policies, monitoring events and resource changes, and provisioning the downstream cluster.

Overall, we believe that Rancher is the best option for managing containerized infrastructures with multiple clusters. In comparison to other multi-cluster alternatives [111], such as Kubefed [112], it offers advantages such as centralized authentication and RBAC for downstream clusters, as well as a superior graphical interface that facilitates user-friendliness. Compared to a single cluster Kubernetes deployment consisting of a master node on the central hub and worker nodes located at the various sites, it provides autonomy at each site, ensuring availability even when the central hub is unavailable.

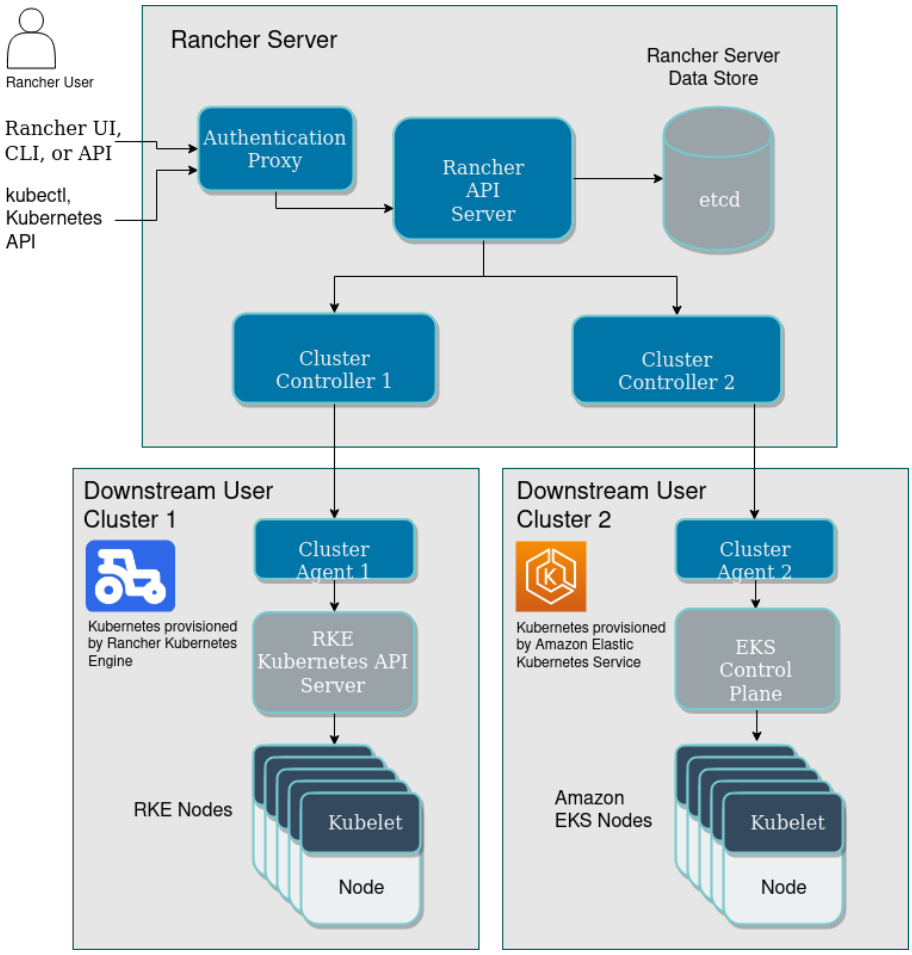


FIGURE 7.2: Rancher architecture [110].

7.3.3 Multi-domain Network Stitching

Even though the multi-domain orchestrator provides the ability to manage multiple containerized clusters deployed on different sites, it is not sufficient to deploy a workflow, or in our case, a 5G experimental network, across these clusters. The missing piece is network stitching that given multiple network subnets as input, will produce a single end-to-end network as output. In this work, we propose that the Submariner framework (Fig. 7.3) [113] assumes the role of network stitching module. Submariner is a sandbox project of the Cloud Native Computing Foundation (CNCF) that offers encrypted or unencrypted L3 cross-cluster connectivity, service exporting and discovery across clusters, and support for the interconnection of clusters with overlapping CIDRs.

Submariner consists of multiple components to ensure connectivity between clusters. The first of these components is the Gateway Engine, which is deployed in each cluster and which primary function is to create tunnels to all other clusters. The Gateway Engine supports three tunneling implementations: a) Libreswan [114], an IPsec VPN protocol; b) WireGuard [115], an additional VPN tunneling protocol; and c) unencrypted tunnels via VXLAN. The second component is the Route Agent, which is deployed on each node of each cluster and is responsible for establishing VXLAN tunnels and routing cross-cluster traffic from the node on which it resides to the node that hosts the Gateway Engine, which will then forward the traffic to the destination cluster. The Broker is a singleton component of Submariner that is

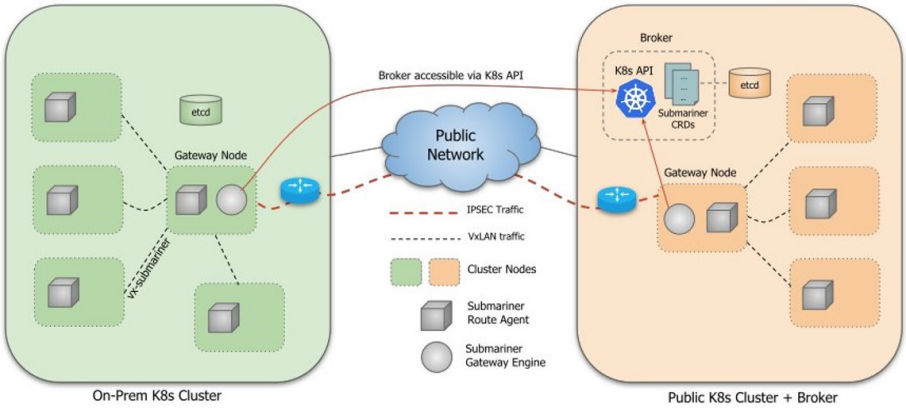
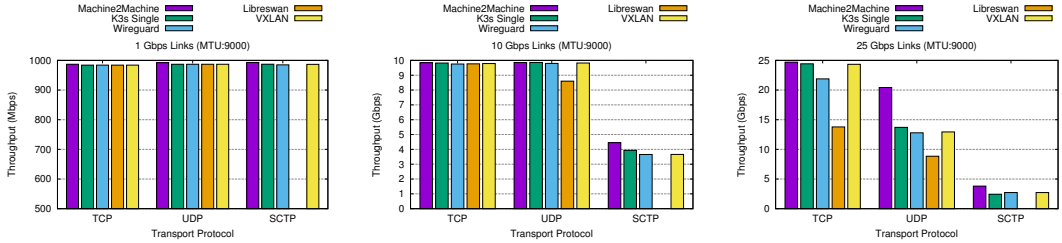


FIGURE 7.3: Submariner network stitching among clusters [113].



(A) Throughput results for 1 Gbps links. (B) Throughput results for 10 Gbps links. (C) Throughput results for 25 Gbps links.

FIGURE 7.4: Throughput performance results under different settings.

deployed on a cluster whose Kubernetes API is accessible by all clusters; therefore, the central hub node is its ideal location for the architecture under consideration. Broker enables the Gateway Engines to discover one another and stores metadata such as the Service and Pod CIDRs for each cluster. It is a central component that enables connectivity between clusters, yet its availability is not always required because Gateway Engines and Route Agents will continue to route traffic based on the most recent information even if it is unavailable. Lighthouse Agent and Lighthouse DNS Server facilitate Service Discovery for Submariner. The Lighthouse Agent is deployed in each cluster and, in coordination with the Broker, imports every service exported by the other clusters. These imports are utilized by the Lighthouse DNS Server to resolve DNS requests received from the Kubernetes cluster's configured CoreDNS service. In the event that a service is deployed in multiple clusters, the Lighthouse DNS server prioritizes the local cluster before selecting a remote cluster in a round-robin fashion. Eventually, an important Submariner component is the Globalnet Controller, which supports overlapping CIDRs in clusters by establishing a virtual network with a global CIDR.

7.4 Evaluation

In this section, we discuss the evaluation results of our proposed framework, which provides multi-domain connectivity and orchestration and enables transparent deployment of 5G experimental networks across distributed testbeds.

To evaluate our approach, we utilize the NITOS testbed [116]. We employ two

single-node clusters based on K3s that are incorporated into a single Rancher instance, each of which runs on a separate node. The specifications of each node are identical, as shown in Table 7.1. We measure and compare the throughput and latency (RTT) obtained in the following cases: a) when using directly the bare metal machines (Machine2Machine), b) when both nodes are integrated in a single K3s cluster (K3s single) that uses Flannel as its default Container Network Interface (CNI), or c) when each node is hosting a separate K3s cluster, and the deployed services are exposed using Submariner, configured with the three different tunneling options (Wireguard, Libreswan, and VXLAN). Notably, the K3s clusters are deployed directly on the bare metal servers and not over a virtualized Openstack infrastructure, which is a common deployment method, because enabling VXLAN tunneling over the Openstack provider network would require explicit configuration.

TABLE 7.1: Configurations used in the experiments

Experiment Parameters	**Value**
Processor	Intel Core i7-11700K @ 3.60GHz
Memory	64GB
NIC (1 Gbps experiments)	Realtek RTL8125 2.5Gbps Ethernet
NIC (10 & 25 Gbps experiments)	Netronome Agilio CX25Gbps SFP28
Operating System	Ubuntu 22.04 server (Linux Kernel v6.4.3)
Rancher Version	v 2.7.3
K3s Version	v1.25.11 +k3s1
Submariner Version	v0.15.2
Traffic Generator	iperf v.3.9

In order to provide a full overview of the performance of the various connectivity options, we obtained the aforementioned metrics for three different link speeds (1 Gbps, 10 Gbps, 25 Gbps) by configuring the NICs appropriately and deactivating the auto-negotiation feature. The MTU of the physical interface was set to 9000 bytes, and since both Flannel and Submariner support auto-MTU configuration, all established tunnels and interfaces were configured to account for the tunneling overhead, with Wireguard being the only exception that required manual configuration. Using iperf3 and the three transport protocols TCP, UDP, and SCTP, throughput was measured, while RTT was determined using ping. We considered evaluating the SCTP protocol, as it is used for signaling traffic transfer in 5G and beyond 5G networks. Similarly, we evaluated the network's latency, as it plays a crucial role in serving uRLLC traffic across clusters. All presented results display the average throughput and RTT measurements as the average of 50 consecutive trials.

Figure 7.4 illustrates our results regarding the attained throughput. When using bare metal servers or the Flannel CNI in a single cluster configuration, the achieved TCP throughput is very close to the maximum allowable transfer rate across all experiment scenarios (1 Gbps, 10 Gbps, and 25 Gbps). Comparing the Submariner tunneling options, we find that the VXLAN cable driver is capable of achieving maximum speed, which is close to the bare metal transferable rate. Wireguard comes close to that value, being capable of achieving a throughput of 22 Gbps. Libreswan is the cable driver with the lowest performance, achieving a maximum of 12 Gbps in the 25 Gbps experiments. In both 1 Gbps and 10 Gbps links, all protocols reach their maximal throughput.

In the case of UDP transfers, where by default 4 KByte buffer size is used, bare metal

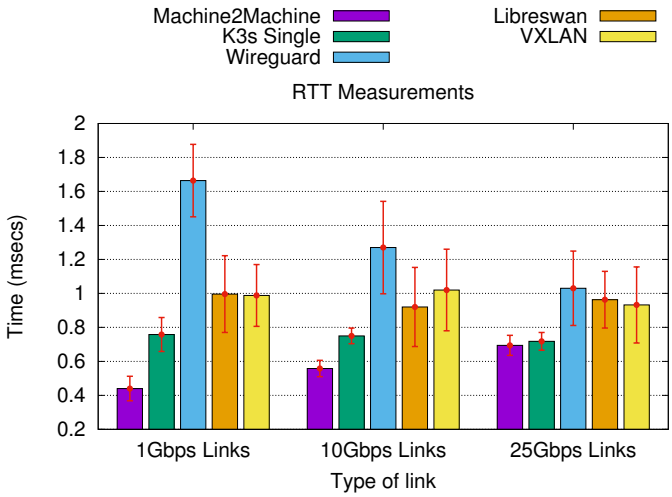


FIGURE 7.5: RTT measurements for the different cable drivers.

and single cluster can attain speeds close to the 10 Gbps maximum rate. However, as the underlying capacity is increased to 25 Gbps, UDP transfer endures high losses, resulting in a maximum of 21 Gbps for the bare metal case and 14.5 Gbps for the single cluster case. Notably, in such high-speed networks, unlike UDP, TCP benefits from the implementations of TCP segmentation and reassembly offloading directly on the card, allowing it to perform better in comparison. VXLAN outperforms the other options in the evaluation of Submariner cable drivers, achieving throughput close to the single cluster case. Wireguard approaches the performance of VXLAN, obtaining a maximum transferable rate of up to 12 Gbps, while Libreswan is the solution with the worst performance, achieving a maximum transferable rate of 9 Gbps.

As our final transport protocol, SCTP with a window size of 4 MBytes is utilized. SCTP achieves a maximum transfer rate of 4.5 Gbps for bare metal transfers, considerably lower than TCP and UDP. When using a single stream for sending/receiving, these rates reflect adjustments to the operation of SCTP, increasing parameters such as the segment size and the maximum surge traffic over the standard implementation. SCTP appears to adapt transfer windows in a less reactive manner than TCP, resulting in a markedly lower throughput. Regarding the various Submariner tunneling options, the performance of VXLAN and Wireguard are comparable. In 25 Gbps links, both cable drivers obtain lower throughput than in 10 Gbps links, 2.7 Gbps and 3.6 Gbps, respectively, a noteworthy observation. Regarding Libreswan, we were unable to transmit SCTP-based traffic because support does not appear to be enabled by default; consequently, the presented results do not indicate any packet transmission.

Figure 7.5 illustrates the average RTT observed across all configurations. Similar to the throughput results, the choice of cable driver affects the overall measured RTT, with Wireguard being the worst performing solution, as indicated by the highest deviation from the average values.

In the overall comparison of the different cable drivers for Submariner, VXLAN and Wireguard seem to be notably outperforming Libreswan. Nevertheless, VXLAN does not implement any encryption on top of the transferable traffic, contrary to the other two solutions. The outcomes are consistent with related benchmarking results [117, 118].

Additionally, the transmission of VXLAN traffic needs to be enabled for cloud-based provisioning based on Openstack or equivalent solutions. The two other cable drivers do not require such configuration because they are transferred over UDP in the case of Wireguard and TCP in the case of Libreswan. As a result, they can be simply implemented on any cloud infrastructure without requiring further modifications. In the case that SCTP traffic needs to be exchanged between deployed workloads (e.g. for the case of control traffic for the 5G/beyond 5G network), Libreswan fails to provide such functionality. In overall, Wireguard appears to be the optimal solution in terms of features and performance, attaining high throughput over encrypted and secure tunnels.

7.5 Conclusion

In this work, we employ an experimentally-driven approach to evaluate Kubernetes-based multi-cluster integration and secure service exposure across different integrated clusters. The solution can be applied as-is for the integration of Kubernetes-based experimental research infrastructures via a central Hub. As indicated by our findings, the selection of the cable driver for the under-study Submariner framework is of the utmost importance, significantly impacting the performance of the workloads deployed on top. Based on the results we obtained, the VXLAN cable driver appears to outperform the other solutions, delivering performance as near as possible to that of the bare metal case. Nonetheless, VXLAN does not provide traffic encryption, hence it is not suitable for use in production-grade environments. Therefore, the next best option is to use the Wireguard driver, which offers encrypted tunnels and comparable performance.

Chapter 8

Optimization of Execution for Machine Learning Applications in the Computing Continuum

Contents

8.1 Introduction 100
8.2 Related Work 101
8.3 MLOps Architecture Overview 101
8.4 Scheduling Algorithm 104
8.5 Evaluation 106
8.6 Conclusion 109

Today, adoption of Machine Learning (ML) techniques is widespread and is encountered in almost every aspect of our everyday lives. The plethora of IoT devices and the enormous amounts of data that are being generated has led to the evolution of existing and the development of new learning algorithms that aim in leveraging data for driving accurate inference and automated decision-making. In parallel, with the computing continuum paradigm, where heterogeneous and distributed resources are stretching all the way from Cloud to Edge and IoT devices, it is imperative that ML applications reap the performance and efficiency benefits of heterogeneity. However, ML application developers and data scientists are faced with the burden of manually deploying their applications in a manner that is frequently sub-optimal.

In this chapter, we present the design and implementation of an integrated MLOps framework that, initially, enables developers to decompose an ML workflow into its functional steps, which correspond to distinct stages of the development and execution of an ML model. Our developed scheduler is therefore able to efficiently schedule these individual components by considering their specific requirements for computing capacity during the training stage, and for low network latency during data ingestion and model serving. The proposed MLOps framework is evaluated with a proof-of-concept experimentation conducted in a realistic testbed environment. Results show significant benefits in performance, when compared with scheduling the whole ML workflow.

8.1 Introduction

Although machine learning (ML) has existed for a number of years, it continues to grow at an exponential rate. Recent advances in ML have been driven by the development of new learning algorithms, the increase in low-cost computing power, and, most importantly, the inexhaustible supply of enormous amounts of data generated by smartphones and IoT devices deployed everywhere. Adoption of ML-based techniques is pervasive in research and industry, leading to the development of ML applications that enable automatic decision making and inference [2] in a wide range of sectors, including healthcare, finance, manufacturing, and agriculture. However, we have yet to witness the full potential and great benefits that this new era will bring to our everyday lives, despite the current heightened interest and hype surrounding AI and ML.

The implementation of ML applications, however, has stringent requirements for data processing and storage, computing and memory capacity, and low-latency responses to inference requests. In the current era, where AI and ML are deployed and consuming resources across the entire computing continuum, from Cloud and Edge to IoT devices, the primary challenge is no longer improving model accuracy, but rather addressing the enormous resource requirements of ML algorithms during their training and model serving. The specific challenge is selecting appropriate resources for the various phases of an ML application's lifecycle, part of MLOps continuous delivery [119], such as model training on performant resources while serving the resulting model in its operational environment in a fast, secure and energy-efficient manner [120]. Currently, considerable effort is required to implement applications on the computing continuum in order to make proper selection and deployment decisions across the vast array of continuum resources.

In this chapter we detail the design and implementation of an integrated framework that allows for the automatic and optimised deployment of a complete ML pipeline on the continuum resources by matching the application needs in terms of computing capacity and response latency with the computing continuum resources. Specifically, the entire ML workflow is described as a pipeline, consisting of the workflow's individual components that are self-contained and containerized pieces of code, each performing a single functional step (e.g. data ingestion, data processing, training, model serving etc). Each component has its own requirements of computing and memory capacity, as well as network latency. Consequently, our framework takes these requirements into account and allows for an optimized and efficient scheduling of the components across the continuum, while also preserving high utilisation and energy efficiency with minimal manual effort from the developers.

The remainder of the chapter is organized as follows: Section 8.2 overviews prior relevant work. Section 8.3 presents the overall architecture of our developed MLOps framework, while section 8.4 discusses the framework's scheduling algorithm. In section 8.5, an evaluation of a small-scale use-case is detailed, and section 8.6, concludes the chapter by summarizing our proposed scheme and outlining future directions for work.

8.2 Related Work

Several research efforts have focused on providing the means for efficiently deploying applications and workloads on the computing continuum that includes Cloud, Fog, Edge and IoT devices. Meng et al. [121] describe a method that dispatches and schedules deadline-aware jobs in edge computing by concurrently considering bandwidth and computing resources. Zhang et al. [122] explore online dispatching and scheduling jobs with various utility functions, whereas Zhou et al. [123] present a scheduler for Distributed Machine Learning that minimizes resource costs and increases social welfare.

Furthermore, there are plenty works that are concerned with ML Model Serving, aiming at reducing inference latency [124], [125]. Similar approaches are mainly focused on reducing computation time through model compression [126] or parallelism [127], while Ogden et al. [128] employ a model selection algorithm to improve accuracy along with on-device duplicate requests to bound their tail response latency. Santos et al. [129] approach the problem of scheduling Fog computing applications by considering delay requirements in their extended Kubernetes scheduler. Our work focuses on providing ML application developers with an MLOps framework that allows them to define the workflow as a pipeline, identifying each step of the process. Consequently, our scheduler considers the entire ML workflow, taking into account the unique requirements of each step, efficiently scheduling it on the computing continuum resources.

8.3 MLOps Architecture Overview

In this section, we present an overview of the architecture of our framework as well as details about the software components we utilize. Our work embraces and extends the workflow concept, which explicitly decouples application-level functionality from deployment, thereby enabling each to be independently optimised. A workflow is constructed by selecting and orchestrating application services to establish a graph of "task steps" that collectively meets the application's demands, and then instantiating each task step by deploying it on a suitable computing continuum resource to fulfill its specific needs.

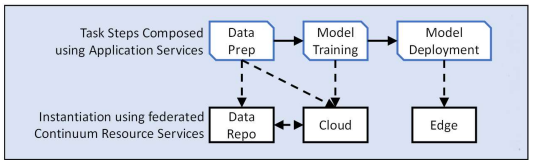


FIGURE 8.1: ML Workflow

8.3.1 ML Workflow

In general, the functional task steps of an ML application can be summarized in the following list:

- **Data Ingestion and Exploration:** This is a crucial phase in the development of the ML model. A data analyst must collect pertinent data to develop the best suitable model for his or her use case.
- **Data Preparation:** Once the data has been retrieved, the data analyst must prepare it. This consists of a) Data Cleaning and Reformatting, b) Data Labeling (in the case of Supervised Learning), and c) Data Splitting between training data used for model training and test data used for evaluation purposes.
- **Model Training:** The training step is the heart of an ML application. Using the training data from the Data Preparation stage, this step trains the model on the specified features. The training process is iterative. The model is trained repeatedly with various subsets of the data until the required degree of accuracy is achieved. This is often a lengthy procedure (including the tuning of hyperparameters) that demands substantial computational resources.
- **Model Evaluation:** In this step, the model is assessed using data never used for training. Several metrics, such as RMSE or AUC-ROC depending on the algorithm type (regression, classification), are used to evaluate the adaptability of a model to unknown data.
- **Model Serving:** In this step, the ML model is deployed in a production environment with stringent accuracy and inference delay constraints.

A subset of such a workflow for an ML application is depicted in Figure 8.1.

8.3.2 Kubeflow

Kubeflow [130] enables the realization of the workflow concept we defined. It is a free and open-source integrated framework that facilitates the orchestration of complex ML workflows that can be deployed on Kubernetes [131]. It comprises of a set of software tools associated with particular ML workflow steps (Jupyter Lab, TensorFlow, TensorBoard, Katib, Seldon Serving, PyTorch serving, etc). Kubeflow Pipeline, which enables the user to express an ML workflow as a simple graph, is the Kubeflow component that we mostly leverage in our framework. Furthermore, Kubeflow offers a dashboard for controlling and monitoring Pipeline execution, in addition to an SDK for defining Pipeline components and a scheduling engine. After the Pipeline and its components are defined, YAML files defining the steps and their relationships are generated. Then, each step is containerized and executed independently within the Kubernetes cluster.

8.3.3 Kubernetes

Kubernetes is an open-source orchestration framework for automating containerized application deployment, administration, and scalability. Kubernetes' architecture is client-server based. It is comprised of a master node and a collection of worker machines, also known as nodes, which execute containerized applications. A multi-master configuration is also feasible, although by default there is a single master node that acts as the "brain" of the cluster. The Master Node provides the control plane for the cluster, making decisions regarding the cluster and responding

to cluster events. It is composed of a number of components that are often installed on the same machine. These components include:

- **Kube-apiserver:** As its name indicates, it is the component that exposes the Kubernetes API and serves as the control plane's front end. Kube-apiserver is meant to horizontally scale in order to evenly distribute traffic.
- **Etcd:** Etcd [54] is a distributed key-value store that serves as the Kubernetes cluster data store (such as number of pods, their desired state, namespace, etc.). It offers resilience and fault-tolerance, and enables the recovery of the Kubernetes cluster in the event of a failure.
- **Kube-scheduler:** Kube-scheduler is responsible for assigning a suitable node to a newly created pod (the smallest working unit in Kubernetes) based on resource utilization and other parameters such as hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.
- **Kube-controller-manager:** Kube-controller-manager is a control plane component that runs and administers controller processes, which are loops that monitor the cluster's status and make or request necessary changes.
- **Cloud-controller-manager:** Cloud-controller-manager interfaces the cluster to the cloud provider's API, therefore isolating components that communicate with the cloud platform from those that interact with the local cluster.

Respectively, the components residing on the worker node are:

- **Kubelet:** The Kubernetes node agent that monitors the cluster's expected and actual states. It starts or stops containers to attain the Kube-apiserver's desired state. It monitors events and resource utilization.
- **Kube-proxy:** It is a network proxy that runs on each cluster node and implements Kubernetes' network service. Kube-proxy maintains network policies on nodes to enable pod connectivity from inside or outside the cluster.
- **Container Runtime:** It runs containers like Docker [132], containerd, CRI-O, and Kubernetes CRI (Container Runtime Interface).

8.3.4 MLOps framework

Our framework combines the fine-grained ML workflow management of Kubeflow with the automated deployment, administration, and scaling capabilities of Kubernetes to allow efficient deployment of the individual workflow steps (Figure 8.2). It accomplishes this by ensuring the fulfillment of the unique requirements of each phase in the development of an ML application (high computing capacities, low latency, etc.) in conjunction with efficient resource management (high utilization, energy efficiency) and the ability to deploy applications on heterogeneous resources in Cloud, Edge, and IoT environments provided by Kubernetes. In further detail, we have enhanced Kubeflow such that ML application developers may label each Pipeline component with the matching name of the ML workflow step. In addition,

a second type of label enables developers to pick one or several locations from a preset list within the Kubernetes cluster to designate from where their model will serve clients.

These labels are then utilized by our extended Kube-scheduler during the scheduling of pods to identify each ML workflow step and efficiently assign it to the nodes that meet the requirements for computing and memory capacity in the case of the training step and low inference latency in the case of the model serving step. In the next section, further details are provided.

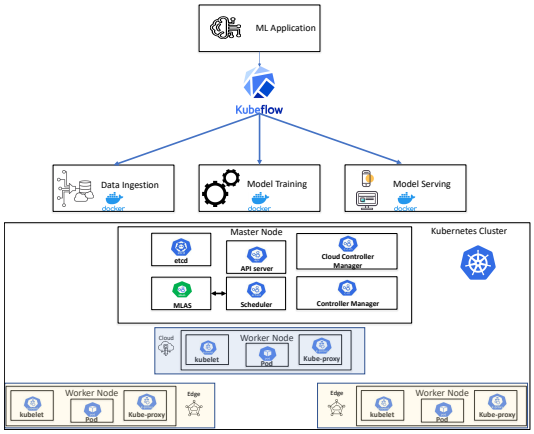


FIGURE 8.2: Overall Architecture

8.4 Scheduling Algorithm

Today, Kubernetes is the standard infrastructure management solution for delivering and managing Cloud, Edge, and IoT applications. The scheduling of the application, i.e. the node(s) on which the application will be deployed, is a vital process for the efficient management of applications. Kubernetes has its own scheduling module, known as Kube-scheduler. This section examines the Kubernetes algorithm behind Kube-scheduler and how it has been modified to accommodate the specific needs of an ML workflow.

8.4.1 Kubernetes Scheduling Algorithm

Kube-scheduler typically operates as an independent component within the Master Node. Kube-scheduler finds an ideal node for each newly created pod or any unscheduled pods to operate on. Nevertheless, each pod and container inside each pod has unique resource needs. To locate the ideal nodes, Kube-scheduler must perform filtering and scoring procedures.

Kube-scheduler continually monitors a queue containing all pods requiring allocation. Then, a two-step procedure is used to pick a node for each pod:

1. **Filtering:** The initial phase is known as filtering. In this stage, the Kube-scheduler verifies whether nodes are capable of executing this pod and eliminates the others. For this reason, it employs certain qualities known as predicates. The **PodFitsResources** element is an example of a predicate. This filter determines if a potential node has sufficient available resources to fulfill a pod's specific resource request. Upon completion of this stage, the node list comprises any acceptable nodes; frequently, there are more.

2. **Scoring:** If the list of appropriate nodes comprises many nodes, the Kube-scheduler will calculate the node priority (Scoring). At this point, Kube-scheduler takes the list of nodes from the filter phase and assigns them a score depending on their priority. The **NodeAffinityPriority** is an example of a priority. The score for the node is determined by node-affinity criteria. For instance, a node with a certain label receives a greater score than others. The node with the highest score is selected to operate the pod.

Kube-scheduler sends the pod to the node with the highest rating. If there are many nodes with identical scores, it picks one at random. Kubelet is then responsible for initiating the pod within the node. Kubernetes enables application developers to define resource limits and requests in the pod configuration file, which supports Kube-scheduler in making better scheduling decisions. The resource limit is the maximum amount of CPU and Memory that may be allotted to containers within a pod. The resource request is the minimum quantity of resources a node should have in order to host this pod.

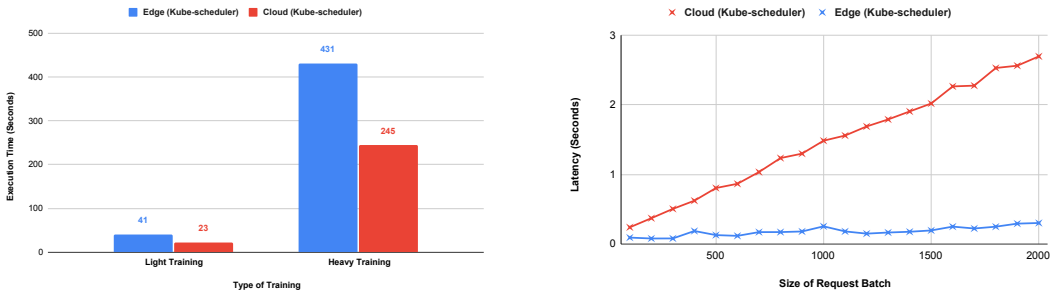
8.4.2 Kubernetes Scheduler Extension

Although Kube-scheduler gives a variety of scheduling options for pods inside the cluster, the decision-making metrics are restricted, as scheduling services make decisions using solely CPU and RAM utilization rates. Kube-scheduler does not consider additional characteristics, such as the application's network latency. However, it is essential for latency-sensitive ML applications to be deployed on nodes with fast inference response times. If this does not occur, the application in question may become unstable and responses may arrive too late. In order to improve the deployment of *Model Serving* and/or *Data Ingestion* steps, we propose in this chapter an extension of the Kubernetes scheduler that makes scheduling decisions based not only on CPU-RAM, but also on latency sensitivity. We called it the **Machine Learning Application Scheduler (MLAS)**.

Regarding implementation, we adopted the approach of implementing a "scheduler-extender," which the Kube-scheduler calls as a final step before making a decision, as this decision is dependent on criteria that are not maintained directly by Kube-scheduler, in our instance, network latency requirements.

8.4.3 Machine Learning Application Scheduler

The MLAS algorithm was implemented by extending Kubernetes' default scheduler via a priority endpoint. This algorithm utilizes Kubernetes Labels and Latency metrics to determine scheduling. Labels are simply key/value pairs that are attached



(A) (B)

FIGURE 8.3: (a) Training Step execution time with Kubernetes Scheduler, (b) Model Serving Step inference latency with Kubernetes Scheduler.

to objects, such as pods, and let users identify important object features. Therefore, in order to identify the attributes that are crucial for our scheduling algorithm, we declare two types of labels for the pod configuration file and one type of label for the node at each location, following a similar approach with [129].

The name of the first label type is **typeOfComponent**. As its name suggests, this label is utilized by the algorithm to identify which step of the ML workflow is being requested to be scheduled. The second type of Label, **Location**, identifies the place from which data is ingested (during the *Data Ingestion* step) or the location of clients requesting model inferences. With the aid of latency values, the MLAS will utilize this Label to deploy the process as near as feasible to this location.

As previously stated, the extended scheduler requires the latency values (as measured by RTT values) of each Location's nodes. So, at each Node, we add a Label for each Location we wish to include in our algorithm, along with the corresponding Latency number. Consider, for instance, Location A and Location B. First, the Latency from the Node to each location is calculated. Let's assume Location A's latency is 0.8 milliseconds and Location B's is 0.6 milliseconds. For each location, the label **{Location : LatencyValue}** is added. In our case, we thus have two Labels:

- {LocationA:0.8ms}
- {LocationB:0.6ms}

The MLAS algorithm initially retrieves the **typeOfComponent** Label from the pod configuration. If the step is neither *Model Serving* nor *Data Ingestion*, the default scheduler determines the appropriate node for scheduling based on its predicates and priorities. If it is the *Model Serving* or *Data Ingestion* step, then it iterates over each node and determines the minimal latency based on the Location given in the pod configuration file at the **Location** Label and returns it to Kubelet for pod deployment.

8.5 Evaluation

In this section, we evaluate our MLOps framework using a small-scale use case involving a real-world ML application. Without focusing on a particular sector, we

TABLE 8.1: Node RTT Values per Location

Node	NITOS-Cloud	NITOS-Edge	Lab-Edge
NITOS-Cloud	0.3 ms	2.7 ms	89.5 ms
NITOS-Edge	2.7 ms	0.5 ms	9.8 ms
Lab-Edge	89.5 ms	9.8 ms	0.6 ms

have selected a generic ML application to demonstrate the benefits on the performance of any application. We deployed our framework and the ML application on the realistic Cloud/Edge environment of the NITOS testbed [116] and assessed performance in terms of training time and inference latency. We compared the scenario in which the application is deployed as a single container in the Cloud or at the Edge without addressing the specific requirements of each phase of the ML workflow to the scenario in which the application is deployed using our MLOps framework that optimizes its execution.

8.5.1 Evaluation Setup

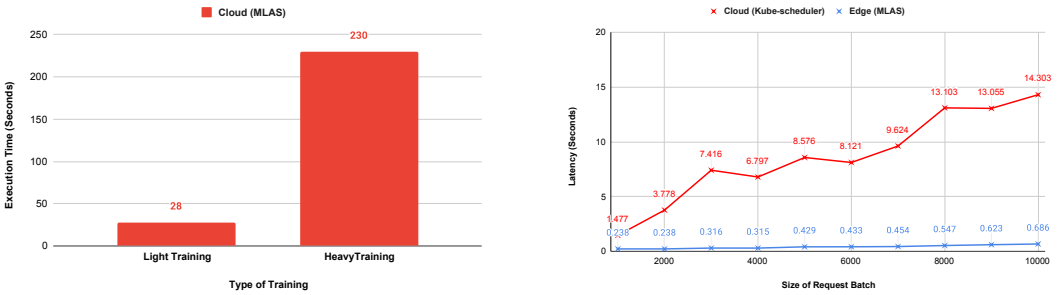
The evaluation was conducted at the NITOS Future Internet Experimentation Facility, which combines 5G, Cloud, Fog, and Edge resources. For our proof-of-concept experiment, we utilized three NITOS nodes in the Cloud facility, one NITOS node at the Edge facility, and one NITOS node at our lab testbed. The respective location names are NITOS-Cloud, NITOS-Edge, and Lab-Edge. All nodes ran Linux Ubuntu 18.04 LTS, and they were all setup with Kubernetes, our scheduler extension, our enhanced version of Kubeflow, Docker, and Calico Network. The cluster had a Master Node hosted in the Cloud and four more nodes functioning as Worker Nodes. As described in Section 8.4, the MLAS algorithm takes response latency (RTT) values into account when scheduling latency-sensitive steps in an ML process. Table 8.1 depicts these values for our evaluation setup.

8.5.2 ML Application

For the proof-of-concept experiment evaluating our MLOps framework, we required an ML application that met the following criteria:

1. The application should be capable of decomposing an ML workflow into its constituent components.
2. The application should include modifiable training parameters that enable us to alter the required processing power.

It turned out that a Kubeflow's example Pipeline was an excellent starting point for showcasing the advantages of our approach. In this example ML application, a regression algorithm is trained in order to predict house prices. We updated the code of this Pipeline to meet our demands and to define steps with specific computing capacity and network latency constraints. In particular, the application ultimately consisted of the following steps:



(A) (B)

FIGURE 8.4: (a) Training Step execution time with MLAS Scheduler, (b) Model Serving Step inference latency comparison between Kubernetes Scheduler and MLAS.

- **Data Ingestion:** Using Kubeflow, data fetching was divided into two independent processes, one for retrieving Training data and the other for retrieving Test data.
- **Data Preparation:** As with the Data Ingestion step, this step is divided into two different processes. It cleans the data by eliminating unnecessary headers and extracts the column of true values from the dataset, which is then utilized in the Model Evaluation step.
- **Model Training:** In this step, the Extreme Gradient Boosting algorithm XGBoost [133] was applied. It is a decision-tree-based ML algorithm that excels with structured/tabular data.
- **Model Evaluation:** It evaluates the model using the true values from the Data Preparation step using the RMSE (Root Mean Square Error) metric.
- **Model Serving:** This step employs the trained model to infer house prices given several inputs such the year it was built, the neighborhood, etc.

8.5.3 Evaluation Results

After installing the software, our MLOps framework, and ML application, we conducted two experiments to compare the performance of the normal practice (single ML application process and use of the default Kubernetes scheduler) with our MLOps framework. Specifically, we measured the Model Training Step's execution time and the Model Serving Step's response time. In the first experiment, the ML application was deployed as a single process, containerized within a single container. The experiment consisted of two scenarios. In the first, the container was deployed on an Edge node, and in the second, on a Cloud node. In both cases, the execution time of a light and a heavy training procedure was measured. As shown in Table 8.2, we varied the "intensity" of training by adjusting the learning rate and number of iterations of the XGBoost algorithm. In addition, we measured the inference latency of Model Serving for request batches of varied sizes. In the second experiment, we deployed the application over our framework, partitioning it into the functional parts of the ML workflow, and using our MLAS scheduler.

TABLE 8.2: Model Training Parameters

Training Intensity	Learning Rate	Num of Iterations	RMSE
Light	0.01	10000	0.075
Heavy	0.001	100000	0.026

Figure 8.3a displays the results for Model Training step execution time and Figure 8.3b the results for Model Serving inference latency. The color blue represents the Edge scenario, whereas the color red represents the Cloud scenario. It is obvious that the execution time for training on Cloud nodes is significantly smaller than on Edge nodes because to the Cloud nodes' greater computational capability. Regarding inference latency, however, we see that the performance disparity between Edge and Cloud resources widens as the batch size increases. Therefore, it is evident that deploying an ML application as a single container on the Cloud or at the Edge cannot provide acceptable training execution duration and model response latency performance.

The second experiment verifies our MLOps framework's effectiveness. As anticipated, the MLAS scheduler schedules the intensive training process on a Cloud node to achieve a quick execution time, whereas the Model Serving step is scheduled on an Edge node based on the latency values indicated in Table 8.1. Figure 8.4a represents the training execution times on the Cloud node, where MLAS scheduled the Model Training step, whereas Figure 8.4b depicts the response latency for varied request batch sizes for the Model Serving step on the Edge node, where it is scheduled. A red line indicates the relevant time if this step was scheduled on a Cloud node for illustrative purposes.

8.6 Conclusion

In this chapter, we detailed the implementation of our MLOps framework, which allows ML application developers and data scientists to deploy their applications in a performance-optimized manner. By integrating and extending orchestration platforms such as Kubeflow and Kubernetes, and by developing our own MLAS scheduler, we were able to decompose an ML application into its functional steps, which we were capable of scheduling efficiently by taking into account their specific requirements. The results of the evaluation validated the performance benefits of our approach. However, future research is necessary to evaluate the system at a larger scale, where many workloads compete for resources, while also taking into account additional performance metrics as well as model complexity that certainly affects inference latency.

Chapter 9

EELAS: Energy Efficient and Latency Aware Scheduling of Cloud-Native ML Workloads

Contents

9.1 Introduction 111
9.2 Related Work 113
9.3 System Model 114
9.4 Evaluation 118
9.5 Conclusion 121

The widespread use of microservices and the use of cloud-native methodologies for the deployment of services have increased the service providers' flexibility and management efficiency. As the available resources for scheduling such workloads have extended the boundaries of traditional datacenters to the fog, edge, and beyond-edge, the scheduling of challenging workloads must also account for energy efficiency, as these devices are typically battery-powered and resource-constrained, while maintaining acceptable performance. Specifically for ML inference workloads, provisioning and access latency plays a crucial role in their successful operation. Towards combating these issues, we design, develop, and evaluate our platform for Energy Efficient Latency-Aware Scheduling (EELAS) of workloads. First, we formulate the scheduling problem as an ILP problem, and then we develop a less complex heuristic method that allows the efficient allocation of resources within the continuum. Our EELAS prototype integrates with Kubernetes and is capable of reducing the overall energy consumption of cloud-to-things resources while accounting for latency of ML workloads. Our evaluation in real-world settings reveals significant energy gains for scheduling ML inference tasks, also reachable with the minimum possible latency from far-edge devices.

9.1 Introduction

Network Functions Virtualization (NFV) has emerged as a key solution for facilitating the deployment of workloads on any kind of underlying hardware. By decoupling the functions of the workloads from the hardware resources, and executing them only in software by making extended use of Virtual Machines or the emerging microservices (e.g. dockers, containers, Unikernels, etc.), the network edges including far edge and fog/mist computing resources can be leveraged to their full

potential. Nevertheless, the complexity and resource consumption of the deployed workloads is constantly rising as they incorporate additional technological capabilities, such as the use of Machine Learning (ML). This, in turn, provides a complex environment for optimal deployment, as devices closer to the network's edge are resource-constrained and reliant on finite batteries to operate. Therefore, scheduling such resource-intensive workloads across the entire cloud-to-things continuum becomes a significant challenge.

As cloud-native deployment of services and workloads becomes the norm, several orchestrators that manage the underlying networking fabric and deploy the workloads on top have emerged. Such solutions include the widely adopted Kubernetes (K8s) framework for the core cloud and the edge, as well as the Rancher K3s platform for resource-constrained edge and beyond edge devices [134]. As a result, none of the existing solutions are able to simultaneously cover deployments over the entire cloud-to-things continuum, taking into consideration aspects such as energy efficiency of the edge/far-edge devices, communication latency, task execution delay, as well as requested resources from the different devices.

In this chapter, we tackle this problem, by appropriately developing extensions to the scheduler of the core Kubernetes framework, towards supporting edge and far-edge devices [135]. Through our extensions, we are able to maximize the overall continuum energy efficiency for deployments of demanding applications, by taking into consideration aspects such as the latency for the deployed services, their location, and their execution time - subject to the underlying host resources that are different as we move from the far-edge to the core cloud. The framework is able to deliver *Energy Efficient and Latency-Aware Scheduling (EELAS)* within the resource continuum. Our solution is tailored especially for the demanding ML inference workloads, and involves continuous monitoring of the available continuum resources, before concluding on the allocation of resources. The solution is provided as a fully-fledged framework, integrated in the K8s environment, and extending existing APIs for orchestration. The main contributions of our work are the following:

- To formulate the allocation problem and provide a heuristic solution.
- To minimize the energy consumption of the cloud-to-things continuum when deploying workloads.
- To maximize the efficiency of ML inference tasks, subject to their latency from the decision making process.
- To develop and evaluate our solution in a fully-fledged framework over a real testbed setup.

We bundle our solution in a real prototype, and evaluate its efficiency over NITOS, the Greek Node of the SLICES-RI [103].

The rest of the chapter is organized as follows: Section 9.2 provides an overview of the related literature. Section 9.3 provides our overall system model, the scheduling problem formulation, as well as our proposed heuristic algorithm. In Section 9.4 we evaluate our solution, and in Section 9.5 we conclude our work and present some future directions.

9.2 Related Work

Service orchestration and scheduling have received a lot of attention since the wide application of NFV architectures. Given the different constraints that the deployed applications and services need to meet, scheduling is not a trivial task, as it must simultaneously fulfill the needs of the infrastructure provider while not violating contracted Service Level Agreements (SLAs) with the end-users. Based on the fact that the CO_2 footprint of datacenters that host such workloads is not negligible, the community has steered its efforts towards energy efficient scheduling. In [136], a classification of the different scheduling algorithms is provided, with a large portion of them being energy efficient methods. Authors in [137] consider solar-powered devices and schedule workloads with the ultimate goal of minimizing the usage of brown energy (the energy of cloud servers) as much as possible. In [138], authors design and evaluate workload schedulers for finding efficient mappings of workflows into resources in order to maximize the quality of service while reducing the energy required to compute them.

As IoT devices are constantly spreading, this creates fertile space for the deployment of services and workloads across a wider resource continuum, as devices are present at the fog and edge levels. In [139], the authors consider a continuum of resources that spans the fog and core cloud. They propose a tree based model for scheduling workloads to the fog nodes and the core cloud, in order to reduce the overall energy consumption. The authors in [140] propose an energy-efficient resource allocation algorithm for the fog computing environment. They use ordered lists to list available resources and schedule the workloads to the least congested, achieving higher efficiency and balancing the load. In [141] authors consider the edge as an environment complementary to the fog and cloud and in real-time enhance the scheduling of data and workloads across the different locations.

As in other fields [2], ML application can prove highly beneficial for the scheduling decisions. Therefore, several works either integrate ML in the scheduling process, or develop energy efficient schedulers for ML tasks, or employ both approaches concurrently. In [142], authors develop their scheme for scheduling ML workloads to different servers with the goal of enhancing the accuracy achieved by the ML tasks. In order to accomplish this, they employ a ML scheme for the scheduler that uses reinforcement learning for updating the scheduler model during operation, and be able to achieve lower execution times and greater accuracy for their workloads. Authors in [143] use a similar approach for making their ML-based scheduler aware of ML workloads that need to be deployed and be able to be retrained during run-time with a reinforcement learning approach. Finally, in [144], authors develop a scheduler aware of possible bottlenecks in the execution of ML workloads, eventually able to schedule/migrate tasks in order to overcome limitations during training.

In this thesis, we extend the existing body of work by developing a scheduling algorithm for ML inference tasks that is energy-efficient and latency-aware. This is one of the first attempts in literature to jointly address the scheduling problem, while considering that the inference tasks need to usually be executed in the most efficient (quickly and energy efficient) manner and as closer to the edge as possible. In the next section, we formulate our problem and the relevant heuristics for reaching the optimal solution.

9.3 System Model

In this section, we define an optimization problem for the energy-efficient scheduling of workloads on heterogeneous cloud, edge, and fog devices in a cloud-to-things continuum cluster, and then formulate it as an Integer Linear Program (ILP). We also present a scalable, energy-efficient scheduling algorithm based on a heuristic approach that reduces complexity and scheduling time.

FIGURE 9.1: The overall EELAS architecture

9.3.1 Problem Definition

We describe the problem of workload scheduling in cloud-to-things continuum resources as follows: Given is a set of resource nodes scattered across the continuum. We assume that the primary factor influencing the power consumption of such a node is the CPU power consumption, which is directly proportional to the load of tasks that are actually executing on the node. As such, we do not consider the power consumption of other components, e.g., memory, storage, and fans. We are provided with the idle and maximum power consumption in Watts per CPU core for each node's CPU module. We are also provided with the Million Instructions Per Second (MIPS) of a CPU core that can be derived through standard benchmarking techniques. Furthermore, we are provided a set of workloads to schedule on the continuum resources, with each workload requiring a minimum QoS in terms of latency. This latency refers to the time needed for the deployed workload, which we consider to be a Machine Learning model, to respond with an inference to a request from a user on a specific location. This time is specified as the execution time of the ML model for a given input, plus the Round Trip Time (RTT) for the request or response to be transferred from or to the user. We assume that the ML workload can be fully parallelized and utilize several CPU cores, and that we are provided with the total number of instructions involved in the execution of the ML workload, which, for simplicity, is independent of the CPU architecture and only depends on processing power.

The objective of the problem is to minimize the total energy consumed by the nodes in the cluster. Because many fog devices are battery-powered, this translates to

longer battery life and thus increased resource availability. However, we are constrained by satisfying the user's requirement for a fast inference response with a minimum acceptable latency. So, scheduling should also take latency into account, not just in terms of network latency between the user and the node where the workload is deployed, but also in terms of how long it takes an ML model to run on a certain node.

9.3.2 Problem Formulation

The definition of the proposed ILP formulation is presented as follows:

Sets:

N	Set of nodes in the cloud-to-things continuum
W	Set of workloads to be deployed
C_n	Set of CPU cores of node $n \in N$
L	Set of locations

Parameters:

P_n^{idle}	Idle power consumption of a CPU core of a node $n \in N$
P_n^{max}	Maximum power consumption of a CPU core of a node $n \in N$
U_{cn}	Utilization of a CPU core $c \in C_n$ of a node $n \in N$ in the range of 0 to 1
IPS_n	Million Instructions Per Second of a CPU core of a node $n \in N$
I_w	Total number of Instructions involved in the execution of a workload $w \in W$
Lat_w	Maximum user-required inference response latency for a workload $w \in W$
$NetLat_{nl}$	Network latency (RTT) between node $n \in N$ and location $l \in L$

Variables:

x_{wn}	Binary variable that equals one if workload $w \in W$ is deployed in node $n \in N$, zero otherwise

The objective is to minimize the total energy consumption in the cluster, given a set of nodes N and workloads to be deployed W.
Objective: Minimize

$$\min \sum_{\mathrm{n} \in \mathrm{N}} E_n \tag{9.1}$$

The above objective function is subject to the following constraints:

$$D_{wn,t} \leq Lat_w, \tag{9.2}$$

where the inference response delay $D_{wn,t}$ of the workload w deployed on a node n at any time slot t must be less or equal to a minimum user-defined QoS latency value. The inference response delay is given

$$D_{wn,t} = T_{wn,t} + NetLat_{nl} \quad \forall \mathrm{w} \in \mathrm{W}, \mathrm{n} \in \mathrm{N}, \mathrm{l} \in \mathrm{L}, \tag{9.3}$$

that equals to the delay from the execution of the ML model w on node n at time slot t, $T_{wn,}$, and the network latency (RTT) between n and the location l of the user, $NetLat_{nl}$.

$$\sum_{\mathrm{n} \in \mathrm{N}} x_{wn} \leq 1 \quad \forall \mathrm{w} \in \mathrm{W} \tag{9.4}$$

$$\sum_{\mathrm{w} \in \mathrm{W}} x_{wn} \leq W_{max} \quad \forall \mathrm{n} \in \mathrm{N} \tag{9.5}$$

Constraint (9.4) limits the amount of nodes on which a workload can be deployed to just one, as we do not consider replication or slicing of the workload. Inequality in the constraint denotes the case where the workload is rejected as it cannot satisfy the constraints, and the sum of x_{wn} equals zero. Additionally, constraint (9.5) limits the amount of workloads that can be deployed on a single node to the maximum value W_{max}. In the case of a K8s system, the parameter W_{max} is equal to 110 workloads (pods) per node.

The execution time of a workload w on node n at the time slot t is provided by

$$T_{wn,t} = x_{wn} \frac{I_w}{Sp_{n,t}} \quad \forall \mathrm{w} \in \mathrm{W}, \mathrm{n} \in \mathrm{N}, \tag{9.6}$$

where $Sp_{n,t}$ is the speed of the CPU on the node n at the same time slot, and which is defined as

$$Sp_{n,t} = |C_n| IPS_n - \tag{9.7}$$
$$\Big(\sum_{\mathrm{c} \in \mathrm{C}_n} U_{cn,t} IPS_n + \sum_{\mathrm{w} \in \mathrm{W}} x_{wn} \frac{I_w}{|C_n| IPS_n} \Big) \quad \forall \mathrm{n} \in \mathrm{N},$$

where the first term refers to the total computing capacity of the CPU on the node n given the number of cores $|C_n|$ and the second to the utilization of cores by already-deployed workloads plus the estimation for the utilization of the ones to be deployed.

The energy consumption of a node n after the scheduling and execution of the workloads W, can be derived by

$$E_n = \sum_{\mathrm{w} \in \mathrm{W}} x_{wn} \sum_{t=0}^{T_{wn}} P_{wn,t}^{cpu} \quad \forall \mathrm{n} \in \mathrm{N}, \tag{9.8}$$

that corresponds to the sum of the node's CPU power consumption for the execution of each workload w deployed on that node. The consumption of the CPU of n for

executing workload w at the time slot t is given by

$$P^{cpu}_{wn,t} = \sum_{c \in C_n} P^{core}_{wcn,t} \quad \forall w \in W, n \in N, \tag{9.9}$$

and corresponds to the total power consumption of the CPU cores for the given workload w on node n. This, in turn, is defined as

$$P^{core}_{wcn,t} = (P^{max}_n - P^{idle}_n)(U_{cn,t} + \frac{I_w}{IPS_n}) + P^{idle}_n \tag{9.10}$$
$$\forall w \in W, c \in C_n, n \in N,$$

and equals to the amount of power consumed by a CPU core c given its utilization by already-deployed workloads at time slot t plus the estimated utilization by workload w. Even when a CPU core is not in use, it consumes power equal to P^{idle}_n.

As it is evident, this problem is not solved in deterministic time in large systems, due to its high-complexity and a heuristic approach is necessary for providing a close-to-optimal solution in a reasonable time.

9.3.3 Heuristic Approach

In this section, we present our algorithm for energy-efficient scheduling with latency constraints, which is based on a greedy heuristic approach [145]. This algorithm was developed in order to achieve better scheduling times in large-scale clusters of nodes, as it is quicker and has less complexity as compared with the ILP. Each iteration begins with a random allocation, and then attempts to improve the energy efficiency by adding and removing nodes from the allocation until a local minimum value is reached for that iteration. The iterations end when there is no longer an improvement to the global minimum value, which is the minimum of all iterations. Before proceeding with the description of the algorithm, we define a metric for the approximate estimation of the energy efficiency of each node, $EnEffIndex_n$, expressed in units of MIPS/Watt.

$$EnEffIndex_n = \frac{IPS_n}{P^{max}_n} \quad \forall n \in N \tag{9.11}$$

Algorithm 2 is the pseudocode of the proposed heuristic scheduler, Energy Efficient and Latency Aware Scheduler (EELAS), that allocates resources for a batch of workloads W across a set of cloud-to-things continuum nodes N. It takes as inputs W and N and sorts the set of workloads W in descending order of the number of instructions required for the execution of each workload, I_w. It is worth noting here that we are assuming that these values are derived from the prior application of benchmarking techniques to each workload. Our rationale behind the sorting is that the workloads with longest execution times, and thus more complexity should be scheduled first, as the shortest ones have a smaller contribution towards the energy efficiency of the system and are therefore easier to schedule. The outer **while** loop in Line 2 executes until there is no improvement in the quality of the solution after k_{max} iterations. An initial random set of nodes S consisting of as many nodes as the number of workloads to be scheduled or the maximum number of nodes in the cluster is generated. In Line 4, we assign workloads to the set of nodes S and obtain a value of energy consumption for this assignment by calling the function *EnergyAssign*.

This function's pseudocode is described by the Algorithm 3. It takes as inputs the sets of workloads W and the set of nodes S, with the latter sorted in descending order of $EnEffIndex$. Then, for each workload w, a node s is assigned, beginning from the most energy efficient, if the inference response delay is less or equal to Lw. Then, the energy consumed value for this node s is updated by adding the estimated energy consumption caused by the execution of the workload w. If w cannot be assigned to any of the nodes, a flag A_w indicating the failure in allocation is returned to the calling function. When all workloads have been assigned to a node, the total energy consumption is then determined and returned.

The EELAS algorithm receives the energy consumption value and keeps it as the minimum current value. It also receives a vector A representing the success or failure of workloads' assignment. Then, within the **repeat-until** loop it updates S by adding and removing nodes from the set of cluster nodes N until it reaches a local energy minimum (i.e. until E_{min} can no longer be decreased) that is stored in E'_{min}. If E_{min} is less than the global energy minimum $Gbal_{min}$, then $Gbal_{min}$ is set to E_{min} and x_{wn} stores the assignment of workloads to nodes, that achieves this minimum. Then a new random set S is generated and the outer **while** loop repeats.

9.4 Evaluation

For the purpose of evaluating EELAS, we have implemented a prototype of our framework and deployed it on a realistic testbed comprising nodes distributed along: 1) the edge, 2) the fog, and 3) the cloud. EELAS is the orchestrator that schedules workloads i.e., ML models for providing inference to users.

In the following part, we describe our experimental setup and prototype's technical information, and in subsection 9.4.2, we show our experimental findings.

9.4.1 Experimental Setup

To evaluate our approach, we employ the NITOS testbed [116], part of the SLICES-RI [103]. As illustrated in Figure 9.1, we employ 3 separate nodes to represent the various nodes that exist along the continuum: a resource-constrained, edge-based device, a compute node for the fog network, and a piece of the cloud infrastructure for the cloud configuration. We setup the nodes in a K8s (version 1.18.18) cluster, which is extended and configured to also include the edge device (Raspberry Pi 4b) and adjust the delays between the nodes to mimic a real network configuration (the RTTs of user devices to edge are $\leq 5ms$, to fog $\leq 20ms$, to cloud $\geq 20ms$). Our framework was implemented by extending the default K8s scheduler with a priority endpoint so that our heuristic approach could be executed during workload scheduling. EELAS retrieves the values of latency requirements, location, and current workload instructions from the K8s YAML configuration files using labels as described in [6].

We use an inference task of object detection from a video file, utilizing OpenVino [146] (version 2020.3) and OpenCV [147] (version 4.6.0) as the ML workload, and we differentiate between the workloads by altering the resolution, and thus the size of the video that we feed as input to the ML model. The execution times on different sorts of nodes for the three video resolutions low (480p), medium (720p), and high (1080p) are listed in the Table 9.2.

Algorithm 2 Energy Efficient and Latency Aware Scheduler

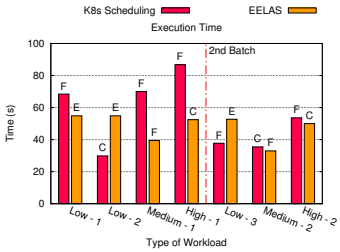
```
1: procedure EELAS(W, N)
2:     while k < k_max do
3:         Generate Random S
4:         E_sol, A_W ← EnergyAssign(W, S)
5:         E_min ← E_sol
6:         repeat
7:             E'_min ← E_min
8:             for n ∈ (N − S) do
9:                 Add n to S
10:                E_sol, A_W ← EnergyAssign(W, S)
11:                if E_sol < E_min and A_W = 1 then
12:                    E_min ← E_sol
13:                else
14:                    Remove n from S
15:                end if
16:            end for
17:            for s ∈ S do
18:                Remove s from S
19:                E_sol, A_W ← EnergyAssign(W, S)
20:                if E_sol < E_min and A_W = 1 then
21:                    E_min ← E_sol
22:                else
23:                    Add s to S
24:                end if
25:            end for
26:            if E_min ≥ Gbal_min then
27:                k ← k + 1
28:            else
29:                k ← 0
30:                Gbal_min ← E_min
31:            end if
32:        until E_min ≥ E'_min
33:    end while
34: end procedure
```

Algorithm 3 Energy Consumption Aware Assignment

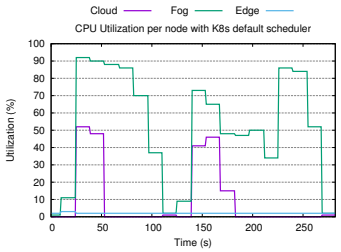
procedure ENERGYASSIGN(W, S)
  Sort S by $EnEffIndex$
  for $w \in W$ **do** ▷ Outter
    for $s \in S$ **do** ▷ Inner
      if $D_{ws} \leq Lw$ **then**
        Assign w to s
        $E_s \leftarrow E_s + \sum_{t=0}^{T_{ws}} P_{ws,t}^{cpu}$
        break ▷ Both loops
      end if
    end for
    if w not assigned **then**
      $A_w \leftarrow 0$
    end if
  end for
  for $s \in S$ **do**
    $E_{sol} \leftarrow \sum_{n \in N} E_n$
  end for
end procedure

TABLE 9.1: Hardware specification

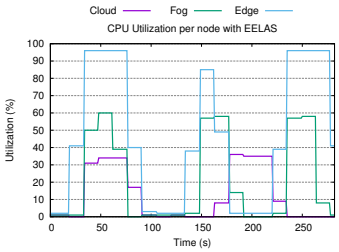
	CPU Model	Cores	RAM	P^{max}	P^{idle}
Cloud	Intel Xeon E-2176g	32	64 GB	114 W	33 W
Fog	Intel Xeon Silver 4215	8	32 GB	85 W	27 W
Edge	Raspberry Pi 4b	4	2 GB	6 W	4 W



(A) Execution time and allocations



(B) Default K8s scheduling



(C) EELAS scheduling

FIGURE 9.2: CPU Utilization per node (Edge/Fog/Cloud) and allocation for the inference workloads

9.4.2 Experiment Evaluation

In order to evaluate the scheduling decisions and consequently the performance of EELAS, we compare our implementation to K8s's default scheduler. We schedule two batches of workloads in the order shown in Figure 9.2a. The letters above the bars denote the node on which each workload was scheduled: C for cloud, F for fog,

and E for edge. K8s scheduler performs best effort scheduling and places the workloads on the fog and cloud nodes. As a result, workload "High-1" of the first batch takes longer to execute due to the increased utilization on that node. This is also evident by the execution time of the similar "High-2" workload of the second batch, deployed on the same node, which is executed in 40% less time. EELAS scheduler prefers to schedule low resolution workloads on the edge node, medium on the fog node, and high on the cloud node. As a result, we observe that execution times in the first batch are lower on average compared to K8s scheduler as workloads are spread across the continuum cluster. Figures 9.2b, 9.2c depict the utilization of the nodes' CPUs during the experiment for the K8s and EELAS schedulers, respectively. We can notice that the resource-constrained Raspberry Pi reaches high utilization values even for low-resolution workloads. Obviously, the fog node also exhibits high utilization values with the default K8s scheduling as most of the workloads are deployed there. Using these utilization values and the power consumption of each node's CPU from Table 9.1, we can obtain the total energy consumed across the cluster using Equation (9.8). Results show that the deployment of workloads using the EELAS scheduler achieves an impressive 41.8% reduction in energy consumption compared to the default K8s scheduler. Moreover, the allocation of workloads to different nodes allows them to finish their execution in less time, thus improving the throughput for inference requests from the users, as the network on the host nodes is usually shared between all the different workloads.

TABLE 9.2: Execution time for each workload on different nodes

	Edge	**Fog**	**Cloud**
480p	29.3s	23.2s	19.6s
720p	40.3s	33.7s	30.2s
1080p	59.7s	52.1s	47.7s

9.5 Conclusion

In this work, we designed, developed, and evaluated our platform for Energy Efficient Latency-Aware Scheduling (EELAS) of ML inference workloads. The problem of allocating the workloads in the cloud-to-things resource continuum was formulated as an ILP, and our developed, less complex heuristic algorithm allows us to find a near-optimal solution in less time. The designed framework was materialized in a testbed environment and was able to achieve an overall energy efficiency of over 41.8% compared to the off-the-shelf K8s scheduler. This work may be extended in several directions. One promising avenue involves supporting the scheduling of training tasks, particularly in the context of federated learning within the resource continuum, as well as deciding on possible migrations of workloads in order to achieve higher efficiency, as well as scaling of the deployed workloads. Moreover, we foresee extending this work with the use of ML learning methods for addressing dynamic profiling of the workloads that are imminent to be deployed when submitted to the EELAS scheduler.

Chapter 10

Conclusions and Future Work

This thesis investigates the gaps and proposes solutions for intelligent and automated orchestration and monitoring across a spectrum of heterogeneous networked systems. The main objective is to develop frameworks and methodologies that enable significant improvements in performance, resilience, and energy efficiency. Each chapter addresses a distinct aspect of this goal, covering domains such as residential WLANs, MANETs, disaggregated datacenters, and the cloud-to-things continuum. Despite the considerable differences among these environments, a consistent challenge persists: how to enable adaptive, efficient, and automated orchestration that accounts for each domain's practical limitations and real-time conditions.

The first two chapters covered the troubleshooting of performance degradation in urban WLAN deployments by leveraging MAC-layer statistics exposed by the wireless card's driver. We developed a user-level framework that, based on these statistics, detects and classifies common 802.11 pathologies such as contention, interference, and the hidden terminal phenomenon. The first chapter established the foundation by thoroughly examining how simple metrics derived from transmission statistics can characterize various pathologies. It also introduced a probing mechanism and a statistical tool for precise diagnosis. Then, the second chapter presented two modes for probing, a passive and an active one, which offer a trade-off between accuracy and overhead, as well as detailed the evaluation of several machine learning algorithms that enable accurate pathology classification. The results were promising and emphasized the importance of vendors exposing MAC-layer statistics to user space. Future work should focus on extending the framework's scope to support multi-label classification, allowing for the diagnosis of multiple coexisting pathologies.

The subsequent two chapters explored the potential of integrating Software Defined Networking in Mobile Ad Hoc Networks. We introduced an SDN-based architecture that included fault-tolerant mechanisms, a cross-layer routing scheme, and distributed topology discovery while maintaining in-band control plane connectivity and thus, compatibility with single-radio wireless nodes. The effectiveness of this approach and the benefits of SDN integration into MANETs were demonstrated through the extensive emulation and proof-of-concept experiments. Based on the developed framework, we then presented the development of a reinforcement learning agent that autonomously adjusts control-plane parameters in response to the underlying network conditions, balancing the trade-off between overhead and responsiveness. The results showed significantly improved performance metrics. As for future research, we propose the use of decentralized, multi-agent RL for improving scalability and the integration of energy-aware metrics into the reward function

to optimize for both performance and device lifetime.

In the datacenter domain, we introduced a software-defined control plane tailored to disaggregated architectures, where compute and memory resources reside on distinct pools that are optically interconnected. The developed SDM Controller orchestrates the dynamic allocation of compute and memory resources by configuring the optical interconnect and coordinating the attachment of remote resources, aiming to minimize scheduling time and energy consumption. Our framework incorporated a graph-based model of resources and a query engine that demonstrated a significant performance gap in comparison to traditional relational databases, as validated by our experimental results. Additionally, the results showed substantial efficiency gains for our scheduler, particularly in large-scale setups. Future research could involve the incorporation of multi-tenant isolation mechanisms in disaggregated resource pools and the introduction of predictive scheduling models.

Following this, we examined the problem of multi-cluster orchestration in (5G) experimental deployments. We presented a Kubernetes-based orchestration framework built on state-of-the-art open-source tools, such as SUSE Rancher and Submariner, for securely interconnecting geographically distributed clusters. This framework enables researchers to deploy and manage workloads across federated 5G testbeds with minimal manual configuration, while maintaining high performance and security. Experimental evaluation over high-speed interconnects revealed that the choice of the underlying cable driver significantly impacts workload performance. In particular, VXLAN provided near bare-metal throughput but lacked encryption, whereas WireGuard offered secure tunneling with only minimal performance overhead. Future work may focus on service discovery optimization, integration with zero-trust models for security, and dynamic workload migration across administrative domains.

The orchestration of ML workflows across the computing continuum was the focus of the concluding chapters. Initially, we created an MLOps framework that facilitates the fine-grained decomposition and scheduling of the different stages of an ML workflow while accounting for the unique compute, latency, and throughput requirements of each stage. Our custom orchestration logic and developed scheduler were built on top of the popular container management platform, Kubernetes, and ML workflow orchestration platform over Kubernetes, Kubeflow. The evaluation demonstrated the system's ability to balance inference latency, training throughput, and resource utilization.

Lastly, we presented EELAS, a framework for energy-efficient and latency-aware scheduling of ML inference workloads. By formulating the workload placement problem as an ILP and developing an effective heuristic, we were able to achieve over 41% energy savings compared to the default Kubernetes scheduling, while at the same time meeting the latency constraints. The results confirmed EELAS's suitability for scheduling inference workloads across the computing continuum, especially for edge and far-edge devices where energy efficiency is crucial. Potential extension of EELAS could involve the implementation of support for scheduling training tasks in the context of federated learning, as well as deciding on potential migrations of already deployed workloads.

Overall, the contributions of this thesis represent a comprehensive effort toward the development of practical, intelligent, autonomous, and adaptive orchestration

frameworks for modern heterogeneous networked systems. Rather than developing a one-size-fits-all approach, this work has developed targeted, context-aware mechanisms that, together with detailed system visibility and intelligent automation, are able to improve performance, energy efficiency, and resilience. Ultimately, this thesis opens multiple pathways for these frameworks and ideas to be extended across even more complex, dynamic, and resource-constrained environments.

Bibliography

[1] I. Syrigos, S. Keranidis, T. Korakis, and C. Dovrolis. "Enabling Wireless Lan Troubleshooting". In: *International Conference on Passive and Active Network Measurement*. Springer. 2015, pp. 318–331.

[2] I. Syrigos, N. Sakellariou, S. Keranidis, and T. Korakis. "On the Employment of Machine Learning Techniques for Troubleshooting WiFi Networks". In: *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. 2019.

[3] I. Syrigos, A. Prassas, I. Koukoulis, K. Choumas, and T. Korakis. "Dynamic SDN Configuration in MANETs: A Reinforcement Learning Approach". Manuscript submitted for publication to *IEEE Access*. 2025.

[4] I. Syrigos, D. Syrivelis, and T. Korakis. "On the Implementation of a Software-Defined Memory Control Plane for Disaggregated Datacenters". In: *2022 IEEE 11th International Conference on Cloud Networking (CloudNet)*. 2022, pp. 177–185.

[5] I. Syrigos, N. Makris, and T. Korakis. "Multi-Cluster Orchestration of 5G Experimental Deployments in Kubernetes over High-Speed Fabric". In: *2023 IEEE Globecom Workshops (GC Wkshps)*. 2023, pp. 1764–1769. DOI: 10.1109/GCWkshps58843.2023.10465054.

[6] I. Syrigos, N. Angelopoulos, and T. Korakis. "Optimization of Execution for Machine Learning Applications in the Computing Continuum". In: *2022 IEEE Conference on Standards for Communications and Networking (CSCN)*.

[7] I. Syrigos, D. Kefalas, N. Makris, and T. Korakis. "EELAS: Energy Efficient and Latency Aware Scheduling of Cloud-Native ML Workloads". In: *2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. 2023, pp. 819–824.

[8] L. Feilu, L. Jian, T. Zhifeng, T. Korakis, E. Erkip, and S. Panwar. "The Hidden Cost of Hidden Terminals". In: *Proc. of ICC*. 2010.

[9] J. Lee et al. "An experimental study on the capture effect in 802.11a networks". In: *Proc. of WinTECH*. 2007.

[10] S. Rayanchu, A Mishra, D. Agrawal, S. Saha, and S. Banerjee. "Diagnosing Wireless Packet Losses in 802.11: Separating Collision from Weak Signal". In: *Proc. of INFOCOM*. 2008.

[11] S. Gollakota, F. Adib, D. Katabi, and S. Seshan. "Clearing the RF Smog: Making 802.11N Robust to Cross-technology Interference". In: *Proc. of SIGCOMM*. 2011.

[12] K. Lakshminarayanan, S. Sapra, S. Seshan, and P. Steenkiste. "RFDump: An Architecture for Monitoring the Wireless Ether". In: *Proc. of CoNEXT*. 2009.

[13] K. Lakshminarayanan, S. Seshan, and P. Steenkiste. "Understanding 802.11 Performance in Heterogeneous Environments". In: *Proc. of HomeNets*. 2011.

[14] P. Kanuparthy, C. Dovrolis, K. Papagiannaki, S. Seshan, and P. Steenkiste. "Can User-Level Probing Detect and Diagnose Common Home-WLAN Pathologies?" In: *ACM SIGCOMM Computer Communication Review* 42.1 (2012), pp. 7–15.

[15] S. Rayanchu, A. Patro, and S. Banerjee. "Airshark: detecting non-WiFi RF devices using commodity WiFi hardware". In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM. 2011, pp. 137–154.

[16] S. Rayanchu, A. Patro, and S. Banerjee. "Catching Whales and Minnows Using WiFiNet: Deconstructing Non-WiFi Interference Using WiFi Hardware." In: *Proc. of NSDI*. 2012.

[17] Y. Cheng, J. Bellardo, P. Benkö, A. Snoeren, G. Voelker, and S. Savage. "Jigsaw: Solving the Puzzle of Enterprise 802.11 Analysis". In: *Proc. of SIGCOMM*. 2006.

[18] R. Chandra, V. Padmanabhan, and M. Zhang. "WiFiProfiler: Cooperative Diagnosis in Wireless LANs". In: *Proc. of MobiSys*. 2006.

[19] A. Sheth, C. Doerr, D. Grunwald, R. Han, and D. Sicker. "MOJO: A Distributed Physical Layer Anomaly Detection System for 802.11 WLANs". In: *Proc. of MobiSys*. 2006.

[20] V. Shrivastava, S. Rayanchu, S. Banerjee, and K. Papagiannaki. "PIE in the Sky: Online Passive Interference Estimation for Enterprise WLANs." In: *Proc. of NSDI*. 2011.

[21] D. Giustiniano, D. Malone, D. Leith, and K. Papagiannaki. "Measuring Transmission Opportunities in 802.11 Links". In: *IEEE/ACM Trans. Netw.* 18.5 (2010).

[22] *"Wireless Chipsets Drivers"*, http://wireless.kernel.org/en/users/Drivers.

[23] J. Jangeun, P. Peddabachagari, and M. Sichitiu. "Theoretical maximum throughput of IEEE 802.11 and its applications". In: *Proc. of NCA*. 2003.

[24] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. "Performance anomaly of 802.11b". In: *Proc. of INFOCOM*. 2003.

[25] G. Bianchi. "Performance analysis of the IEEE 802.11 distributed coordination function". In: *IEEE JSAC* 18.3 (2000), pp. 535–547.

[26] V. Cisco. *Cisco Visual Networking Index: Forecast and Methodology 2016–2021.(2017)*. 2017.

[27] C. Fortuna, E. De Poorter, P. Škraba, and I. Moerman. "Data Driven Wireless Network Design: A Multi-level Modeling Approach". In: *Wireless Personal Communications* 88.1 (2016), pp. 63–77.

[28] K.-H. Kim, H. Nam, and H. Schulzrinne. "WiSlow: A Wi-Fi Network Performance Troubleshooting Tool for End Users". In: *INFOCOM, 2014 Proceedings IEEE*. IEEE. 2014, pp. 862–870.

[29] K. Sui et al. "Characterizing and Improving WiFi Latency in Large-Scale Operational Networks". In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM. 2016, pp. 347–360.

[30] M. O. Khan and L. Qiu. "Accurate WiFi packet delivery rate estimation and applications". In: *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE. 2016, pp. 1–9.

[31] U. Paul, A. Kashyap, R. Maheshwari, and S. R. Das. "Passive Measurement of Interference in WiFi Networks with Application in Misbehavior Detection". In: *IEEE transactions on mobile computing* 12.3 (2013), pp. 434–446.

[32] N. Inzerillo, D. Croce, D. Garlisi, F. Giuliano, and I. Tinnirello. "Error-Based Interference Detection in WiFi Networks". In: *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE. 2017, pp. 1–6.

[33] K. Chounos, P. Karamichailidis, N. Makris, and T. Korakis. "Unlicensed Spectrum Forecasting: An Interference Umbrella Based on Channel Analysis and Machine Learning". In: *IEEE Transactions on Network Science and Engineering* 9.5 (2022), pp. 3421–3436.

[34] A. Hithnawi, H. Shafagh, and S. Duquennoy. "TIIM: technology-independent interference mitigation for low-power wireless networks". In: *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*. ACM. 2015, pp. 1–12.

[35] F. Hermans, L.-Å. Larzon, O. Rensfelt, and P. Gunningberg. "A lightweight approach to online detection and classification of interference in 802.15. 4-based sensor networks". In: *ACM SIGBED Review* 9.3 (2012), pp. 11–20.

[36] F. Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[37] MarketsandMarkets. "Software-Defined Networking Market by Component (SDN Infrastructure, Software, and Services), SDN Type (Open SDN, SDN via Overlay, and SDN via API), End User, Organization Size, Enterprise Vertical, and Region - Global Forecast to 2025". In: (2020).

[38] T. Clausen and P. Jacquet. *Optimized link state routing protocol (OLSR)*. Tech. rep. 2003.

[39] A. Detti, C. Pisa, S. Salsano, and N. Blefari-Melazzi. "Wireless Mesh Software Defined Networks (wmSDN)". In: *2013 IEEE 9th international conference on wireless and mobile computing, networking and communications (WiMob)*. 2013.

[40] M. Labraoui, M. Boc, and A. Fladenmuller. "Self-configuration mechanisms for SDN deployment in Wireless Mesh Networks". In: *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 2017.

[41] K. Poularakis, Q. Qin, E. M. Nahum, M. Rio, and L. Tassiulas. "Flexible SDN control in tactical ad hoc networks". In: *Ad Hoc Networks* 85 (2019), pp. 71–80.

[42] D. Giatsios, K. Choumas, P. Flegkas, T. Korakis, J. J. A. Cruelles, and D. C. Mur. "Design and Evaluation of a Hierarchical SDN Control Plane for 5G Transport Networks". In: *2019 IEEE International Conference on Communications (ICC)*. 2019.

[43] H. C. Yu, G. Quer, and R. R. Rao. "Wireless SDN mobile ad hoc network: From theory to practice". In: *2017 IEEE International Conference on Communications (ICC)*. 2017.

[44] P. Bellavista, A. Dolci, and C. Giannelli. "MANET-Oriented SDN: motivations, challenges, and a solution prototype". In: *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. 2018.

[45] K. Streit, N. Rodday, F. Steuber, C. Schmitt, and G. D. Rodosek. "Wireless SDN for highly utilized MANETs". In: *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2019.

[46] B. Pfaff et al. "The Design and Implementation of Open {vSwitch}". In: *12th USENIX symposium on networked systems design and implementation (NSDI 15)*. 2015.

[47] D. Ongaro and J. Ousterhout. "In Search of an Understandable Consensus Algorithm". In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, June 2014. ISBN: 978-1-931971-10-2. URL: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

[48] M. Rademacher, F. Siebertz, M. Schlebusch, and K. Jonas. "Experiments with OpenFlow and IEEE802.11 Point-to-Point Links in a WMN". In: *ICWMC 2016* (2016), p. 111.

[49] S. Sharma, A. Nag, P. Stynes, and M. Nekovee. "Automatic Configuration of OpenFlow in Wireless Mobile Ad hoc Networks". In: *2019 International Conference on High Performance Computing & Simulation (HPCS)*. 2019.

[50] Open Networking Foundation. *OpenFlow Switch Specification*. Tech. rep. Version 1.3.1. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.1.pdf. Open Networking Foundation, Apr. 2013.

[51] X. Chen, T. Wu, G. Sun, and H. Yu. "Software-defined MANET swarm for mobile monitoring in hydropower plants". In: *IEEE Access* 7 (2019), pp. 152243–152257.

[52] D. N. da Hora, K. Van Doorselaer, K. Van Oost, R. Teixeira, and C. Diot. "Passive Wi-Fi Link Capacity Estimation on Commodity Access Points". In: *Traffic Monitoring and Analysis Workshop (TMA) 2016*. 2016.

[53] *Ryu Controller*. URL: https://ryu.readthedocs.io (visited on 10/28/2022).

[54] *Etcd*. URL: https://etcd.io/ (visited on 10/28/2022).

[55] H Rogge and E Baccelli. *Directional Airtime Metric Based on Packet Sequence Numbers for Optimized Link State Routing Version 2 (OLSRv2)*. Tech. rep. 2016.

[56] I. Syrigos, I. Koukoulis, A. Prassas, K. Choumas, and T. Korakis. "On the Implementation of a Cross-Layer SDN Architecture for 802.11 MANETs". In: *2023 IEEE International Conference on Communications (ICC): Mobile and Wireless Networks Symposium (IEEE ICC'23 - MWN Symposium)*. Rome, Italy, May 2023.

[57] D. Kafetzis, S. Vassilaras, G. Vardoulias, and I. Koutsopoulos. "Software-Defined Networking Meets Software-Defined Radio in Mobile ad hoc Networks: State of the Art and Future Directions". In: *IEEE Access* 10 (2022), pp. 9989–10014.

[58] D. Sousa, S. Sargento, and M. Luís. "Hybrid Wireless Network with SDN and Legacy Devices in ad-hoc Environments". In: *2022 13th International Conference on Network of the Future (NoF)*. 2022.

[59] A. Dusia and A. S. Sethi. "Software-Defined Architecture for Infrastructureless Mobile Ad Hoc Networks". In: *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2021.

[60] N. C. Luong et al. "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey". In: *IEEE Communications Surveys & Tutorials* 21.4 (2019), pp. 3133–3174.

[61] R. Amin, E. Rojas, A. Aqdus, S. Ramzan, D. Casillas-Perez, and J. M. Arco. "A Survey on Machine Learning Techniques for Routing Optimization in SDN". In: *IEEE Access* 9 (2021), pp. 104582–104611.

[62] K. Leevangtou, H. Ochiai, and C. Aswakul. "Application of Q-Learning in Routing of Software-Defined Wireless Mesh Network". In: *IEEJ Transactions on Electrical and Electronic Engineering* 17.3 (2022), pp. 387–397.

[63] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca. "Intelligent Routing Based on Reinforcement Learning for Software-Defined Networking". In: *IEEE Transactions on Network and Service Management* 18.1 (2021), pp. 870–881.

[64] Z. Zhang, L. Ma, K. Poularakis, K. K. Leung, J. Tucker, and A. Swami. "MACS: Deep Reinforcement Learning based SDN Controller Synchronization Policy Design". In: *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. 2019.

[65] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo. "QoS-Aware Adaptive Routing in Multi-layer Hierarchical Software Defined Networks: A Reinforcement Learning Approach". In: *2016 IEEE International Conference on Services Computing (SCC)*. 2016.

[66] Y.-R. Chen, A. Rezapour, W.-G. Tzeng, and S.-C. Tsai. "RL-Routing: An SDN Routing Algorithm Based on Deep Reinforcement Learning". In: *IEEE Transactions on Network Science and Engineering* 7.4 (2020), pp. 3185–3199.

[67] Z. Zhang, L. Ma, K. Poularakis, K. K. Leung, and L. Wu. "DQ Scheduler: Deep Reinforcement Learning Based Controller Synchronization in Distributed SDN". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 2019.

[68] E. F. Castillo, O. M. C. Rendon, A. Ordonez, and L. Z. Granville. "IPro: An approach for intelligent SDN monitoring". In: *Computer Networks* 170 (2020), p. 107108.

[69] L. Ochoa Aday, C. Cervelló Pastor, and A. Fernández Fernández. "Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks". In: (2015).

[70] R. Wazirali, R. Ahmad, and S. Alhiyari. "SDN-OpenFlow Topology Discovery: An Overview of Performance Issues". In: *Applied Sciences* 11.15 (2021). ISSN: 2076-3417. URL: https://www.mdpi.com/2076-3417/11/15/6999.

[71] C. Fluri, D. Melnyk, and R. Wattenhofer. "Improving Raft When There Are Failures". In: *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*. 2018.

[72] R. Hanmer, L. Jagadeesan, V. Mendiratta, and H. Zhang. "Friend or Foe: Strong Consistency vs. Overload in High-Availability Distributed Systems and SDN". In: *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 2018.

[73] *EMANE*. URL: https://adjacentlink.com/documentation/emane/v1.0.1/ (visited on 07/25/2023).

[74] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, and M. Schwamborn. "Bonnmotion: a mobility scenario generation and analysis tool". In: *3rd International ICST Conference on Simulation Tools and Techniques*. 2010.

[75] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[76] D. Ernst, P. Geurts, and L. Wehenkel. "Tree-Based Batch Mode Reinforcement Learning". In: *Journal of Machine Learning Research* 6 (2005).

[77] S. Levine, A. Kumar, G. Tucker, and J. Fu. "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: *arXiv preprint arXiv:2005.01643* (2020).

[78] S. Fujimoto, D. Meger, and D. Precup. "Off-Policy Deep Reinforcement Learning without Exploration". In: *International Conference on Machine Learning*. PMLR. 2019.

[79] R. Agarwal, D. Schuurmans, and M. Norouzi. "An Optimistic Perspective on Offline Reinforcement Learning". In: *International Conference on Machine Learning*. PMLR. 2020.

[80] A. Kumar, A. Zhou, G. Tucker, and S. Levine. "Conservative Q-Learning for Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1179–1191.

[81] H. Van Hasselt, A. Guez, and D. Silver. "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2016.

[82] M. Tirmazi et al. "Borg: the next generation". In: *Proceedings of the Fifteenth European Conference on Computer Systems*. 2020, pp. 1–14.

[83] M. Bielski et al. "dReDBox: Materializing a full-stack rack-scale system prototype of a next-generation disaggregated datacenter". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 1093–1098.

[84] D. Syrivelis et al. "A software-defined architecture and prototype for disaggregated memory rack scale systems". In: *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE. 2017, pp. 300–307.

[85] N. Alachiotis et al. "dReDBox: A Disaggregated Architectural Perspective for Data Centers". In: *Hardware Accelerators in Data Centers*. Springer, 2019, pp. 35–56.

[86] A. D. Papaioannou, R. Nejabati, and D. Simeonidou. "The Benefits of a Disaggregated Data Centre: A Resource Allocation Approach". In: *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2016, pp. 1–7.

[87] G. Farias, F. Brasileiro, R. Lopes, M. Carvalho, F. Morais, and D. Turull. "On the Efficiency Gains of Using Disaggregated Hardware to Build Warehouse-Scale Clusters". In: *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2017, pp. 239–246.

[88] H. M. M. Ali, T. E. El-Gorashi, A. Q. Lawey, and J. M. Elmirghani. "Future Energy Efficient Data Centers With Disaggregated Servers". In: *Journal of Lightwave Technology* 35.24 (2017), pp. 5361–5380.

[89] G. Panwar et al. "Quantifying Memory Underutilization in HPC Systems and Using it to Improve Performance via Architecture Support". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 2019, pp. 821–835.

[90] I. Peng, R. Pearce, and M. Gokhale. "On the Memory Underutilization: Exploring Disaggregated Memory on HPC Systems". In: *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE. 2020, pp. 183–190.

[91] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker. "Network support for resource disaggregation in next-generation datacenters". In: *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. 2013, pp. 1–7.

[92] P. X. Gao et al. "Network Requirements for Resource Disaggregation". In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016, pp. 249–264.

[93] A. Roozbeh et al. "Software-Defined "Hardware" Infrastructures: A Survey on Enabling Technologies and Open Research Directions". In: *IEEE Communications Surveys & Tutorials* 20.3 (2018), pp. 2454–2485.

[94] A. Lagar-Cavilla et al. "Software-Defined Far Memory in Warehouse-Scale Computers". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019, pp. 317–330.

[95] N. Nordlund, V. Vassiliadis, M. Gazzetti, D. Syrivelis, and L. Tassiulas. "Energy-Aware Learning Agent (EALA) for Disaggregated Cloud Scheduling". In: *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE. 2021, pp. 578–583.

[96] A. Call, J. Polo, and D. Carrera. "Workload-Aware Placement Strategies to Leverage Disaggregated Resources in the Datacenter". In: *IEEE Systems Journal* (2021).

[97] *JanusGraph*. URL: https://janusgraph.org/ (visited on 03/28/2022).

[98] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins. "A comparison of a graph database and a relational database: a data provenance perspective". In: *Proceedings of the 48th annual Southeast regional conference*. 2010, pp. 1–6.

[99] *Apache TinkerPop*. URL: https://tinkerpop.apache.org/ (visited on 03/28/2022).

[100] *PL/pgSQL*. URL: https://www.postgresql.org/ (visited on 03/28/2022).

[101] Y. Cheng, P. Ding, T. Wang, W. Lu, and X. Du. "Which Category Is Better: Benchmarking Relational and Graph Database Management Systems". In: *Data Science and Engineering* 4.4 (2019), pp. 309–322.

[102] M. Luksa. *Kubernetes in action*. Simon and Schuster, 2017.

[103] S. Fdida et al. "SLICES, a scientific instrument for the networking community". In: *Computer Communications* 193 (2022), pp. 189–203. ISSN: 0140-3664.

[104] F. Kaltenberger, G. d. Souza, R. Knopp, and H. Wang. "The OpenAirInterface 5G New Radio Implementation: Current Status and Roadmap". In: *WSA 2019; 23rd International ITG Workshop on Smart Antennas*. 2019.

[105] srsran-project. *SRS*. [Online], https://github.com/srsran/srsran_project. 2023.

[106] F. J. De Souza Neto, E. Amatucci, N. A. Nassif, and P. A. Marques Farias. "Analysis for Comparison of Framework for 5G Core Implementation". In: *2021 International Conference on Information Science and Communications Technologies (ICISCT)*. 2021. DOI: 10.1109/ICISCT52966.2021.9670414.

[107] M. Ersue. "ETSI NFV management and orchestration-An overview". In: *Presentation at the IETF* 88 (2013).

[108] G. M. Yilma, Z. F. Yousaf, V. Sciancalepore, and X. Costa-Perez. "Benchmarking open source NFV MANO systems: OSM and ONAP". In: *Computer Communications* 161 (2020), pp. 86–98. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2020.07.013. URL: https://www.sciencedirect.com/science/article/pii/S0140366420305946.

[109] M. Amaral. "KubeVirt scale test by creating 400 VMIs on a single node". In: *Free and Open source Software Developers' European Meeting*. 2022.

[110] SUSE. *Rancher*. [Online], https://www.rancher.com/. 2023.

[111] L. Osmani, T. Kauppinen, M. Komu, and S. Tarkoma. "Multi-Cloud Connectivity for Kubernetes in 5G Networks". In: *IEEE Communications Magazine* 59.10 (2021), pp. 42–47. DOI: 10.1109/MCOM.110.2100124.

[112] F. Faticanti, D. Santoro, S. Cretti, and D. Siracusa. "An Application of Kubernetes Cluster Federation in Fog Computing". In: *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. 2021. DOI: 10.1109/ICIN51074.2021.9385548.

[113] Linux CNCF. *Submariner*. [Online], https://submariner.io/. 2023.

[114] Libreswan. *Libreswan - VPN software*. [Online], https://libreswan.org.

[115] Wireguard. *Wireguard - Fast, Modern, Secure VPN tunnel*. [Online], https://www.wireguard.com/.

[116] N. Makris, C. Zarafetas, S. Kechagias, T. Korakis, I. Seskar, and L. Tassiulas. "Enabling open access to LTE network components; the NITOS testbed paradigm". In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. 2015.

[117] P. G. Kannan, B. Salisbury, P. Kodeswaran, and S. Sen. "Benchmarking tunnel and encryption methodologies in cloud environments". In: *arXiv preprint arXiv:2203.02142* (2022).

[118] L. Osswald, M. Haeberle, and M. Menth. "Performance comparison of VPN solutions". In: (2020).

[119] S. Mäkinen, H. Skogström, E. Laaksonen, and T. Mikkonen. "Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help?" In: *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. 2021.

[120] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi. "A Unified Model for the Mobile-Edge-Cloud Continuum". In: *ACM Transactions on Internet Technology (TOIT)* 19.2 (2019), pp. 1–21.

[121] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li. "Online Deadline-Aware Task Dispatching and Scheduling in Edge Computing". In: *IEEE Transactions on Parallel and Distributed Systems* 31.6 (2019), pp. 1270–1286.

[122] C. Zhang et al. "Online dispatching and scheduling of jobs with heterogeneous utilities in edge computing". In: *Proceedings of the Twenty-First International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*. 2020.

[123] R. Zhou, N. Wang, Y. Huang, J. Pang, and H. Chen. "DPS: Dynamic Pricing and Scheduling for Distributed Machine Learning Jobs in Edge-Cloud Networks". In: *IEEE Transactions on Mobile Computing* (2022).

[124] F. Yan, Y. He, O. Ruwase, and E. Smirni. "Efficient Deep Neural Network Serving: Fast and Furious". In: *IEEE Transactions on Network and Service Management* 15.1 (2018), pp. 112–126.

[125] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica. "Clipper: A {Low-Latency} Online Prediction Serving System". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017, pp. 613–627.

[126] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[127] H. Qin, S. Zawad, Y. Zhou, S. Padhi, L. Yang, and F. Yan. "Reinforcement-Learning-Empowered MLaaS Scheduling for Serving Intelligent Internet of Things". In: *IEEE Internet of Things Journal* 7.7 (2020), pp. 6325–6337.

[128] S. S. Ogden and T. Guo. "MDINFERENCE: Balancing Inference Accuracy and Latency for Mobile Applications". In: *2020 IEEE International Conference on Cloud Engineering (IC2E)*. 2020.

[129] J. Santos, T. Wauters, B. Volckaert, and F. De Turck. "Towards Network-Aware Resource Provisioning in Kubernetes for Fog Computing Applications". In: *2019 IEEE Conference on Network Softwarization (NetSoft)*. 2019.

[130] *KubeFlow*. URL: https://www.kubeflow.org/ (visited on 03/28/2022).

[131] *Kubernetes*. URL: https://kubernetes.io/ (visited on 03/28/2022).

[132] *Docker*. URL: https://www.docker.com/ (visited on 03/28/2022).

[133] *XGBoost*. URL: https://xgboost.ai/ (visited on 03/28/2022).

[134] A. Valantasis, N. Makris, and T. Korakis. "Orchestration Software for Resource Constrained Datacenters: an Experimental Evaluation". In: *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*. 2022, pp. 121–126.

[135] Y. Wu. "Cloud-Edge Orchestration for the Internet of Things: Architecture and AI-Powered Data Processing". In: *IEEE Internet of Things Journal* 8.16 (2021), pp. 12792–12805.

[136] G. L. Stavrinides and H. D. Karatza. "Scheduling data-intensive workloads in large-scale distributed systems: trends and challenges". In: *Modeling and simulation in HPC and cloud systems* (2018), pp. 19–43.

[137] I. De Courchelle, T. Guérout, G. Da Costa, T. Monteil, and Y. Labit. "Green energy efficient scheduling management". In: *Simulation Modelling Practice and Theory* 93 (2019). Modeling and Simulation of Cloud Computing and Big Data, pp. 208–232. ISSN: 1569-190X.

[138] B. Dorronsoro, S. Nesmachnow, J. Taheri, A. Y. Zomaya, E.-G. Talbi, and P. Bouvry. "A hierarchical approach for energy-efficient scheduling of large workloads in multicore distributed systems". In: *Sustainable Computing: Informatics and Systems* 4.4 (2014). Special Issue on Energy Aware Resource Management and Scheduling (EARMS), pp. 252–261. ISSN: 2210-5379.

[139] "An energy-efficient model for fog computing in the Internet of Things (IoT)". In: *Internet of Things* 1-2 (2018), pp. 14–26. ISSN: 2542-6605.

[140] A. U. Rehman et al. "Dynamic Energy Efficient Resource Allocation Strategy for Load Balancing in Fog Environment". In: *IEEE Access* 8 (2020).

[141] H. A. Alharbi and M. Aldossary. "Energy-Efficient Edge-Fog-Cloud Architecture for IoT-Based Smart Agriculture Environment". In: *IEEE Access* 9 (2021).

[142] H. Wang, Z. Liu, and H. Shen. "Job Scheduling for Large-Scale Machine Learning Clusters". In: *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '20. Association for Computing Machinery, 2020, 108–120. ISBN: 9781450379489.

[143] Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Meng, and W. Lin. "DL2: A Deep Learning-Driven Scheduler for Deep Learning Clusters". In: *IEEE Transactions on Parallel and Distributed Systems* 32.8 (2021), pp. 1947–1960.

[144] T. Barreto Goes Perez, X. Zhou, L. Liu, and Z. Ding. "Bottleneck-Aware Task Scheduling Based on Per-Stage and Multi-ML Profiling". In: *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2019, pp. 510–518.

[145] P. Lindberg, J. Leingang, D. Lysaker, S. U. Khan, and J. Li. "Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems". In: *The Journal of Supercomputing* 59.1 (2012), pp. 323–360.

[146] Y. Gorbachev, M. Fedorov, I. Slavutin, A. Tugarev, M. Fatekhov, and Y. Tarkan. "Openvino deep learning workbench: Comprehensive analysis and tuning of neural networks inference". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 2019.

[147] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

1

[1]The LaTeX template can be found at the following link.