UNIVERSITÀ DI TRENTO

UNIVERSITY OF TRENTO

DOCTORAL THESIS

Towards Network Automation: A Multi-Agent Based Intelligent Networking System

Author:
Sisay Tadesse ARZO

Supervisor:
Prof. Fabrizio GRANELLI

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

in the

Next Generation Networking Lab
Department of Information Engineering and Computer Science

December 3, 2021

Declaration of Authorship

I, Sisay Tadesse ARZO, declare that this thesis titled, "Towards Network Automation: A Multi-Agent Based Intelligent Networking System" 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: ____

“We are reminded that, in the fleeting time we have on this Earth, what matters is not wealth, or status, or power, or fame, but rather how well we have loved and what small part we have played in making the lives of other people better.” Barack Obama

“We are slowed down sound and light waves, a walking bundle of frequencies tuned into the cosmos. We are souls dressed up in sacred biochemical garments and our bodies are the instruments through which our souls play their music.” Albert Einstein

Social/altruistic Vs biological/physical perspectives of humans. We need both!

Abstract

Sisay Tadesse ARZO

Towards Network Automation: A Multi-Agent Based Intelligent Networking System

This P.h.D thesis has three parts. The first part is the mathematical modeling of softwarized network. We studied network softwarzation through Virtual Network Function (VNF) placement considering 5G and B5G's stringent requirements of latency, reliability, and support for heterogeneous devices. Since the existing wireless network architecture is limited to fulfill these constraints, a cloud radio access network (C-RAN), along with network function virtualization, is suggested to provide flexibility and network agility. C-RAN decouples network functions, such as firewall and packet gateway, from hardware to software deployed in the cloud. Thus, a comprehensive end-to-end formulation of this architecture is required for VNF placement. Most of the existing works focus on virtual function placement with different objectives, addressing different service requirements separately. Six 5G constraints are considered simultaneously to find optimal VNF placement with service differentiation. The selected six parameters reflect services' requirements, network constraints, and computing constraints. We first model the overall cloud radio access network as a multi-layer loopless-random hypergraph and we provide the overall formulation of the system. Then, we reformulate such a model considering backup virtual functions and CPU over-provisioning techniques to improve both virtual function's reliability and processing latency. Finally, we propose service differentiation to reduce CPU utilization and energy consumption, while using the above techniques. The results suggest that the application of service differentiation can significantly improve the assignment of computing resources and energy efficiency. We have also demonstrated a VNF design for IoT interoperability developing a translator as a VNF. Furthermore, we have tested the emulation of LTE, LTE-A, and 5G over a lightweight open platform considering delay analysis.

The second part of the work focus on network automation. The advent of network softwarization is enabling multiple innovative solutions through software-defined networking (SDN) and network function virtualization (NFV). Specifically, network softwarization paves the way for autonomic and intelligent networking, which has gained popularity in the research community. Along with the arrival of 5G and beyond, which interconnects billions of devices, the complexity of network management is significantly increasing both investments and operational costs. Autonomic networking is the creation of self-organizing, self-managing, and self-protecting networks, to manage complex and heterogeneous networks. To achieve full network automation, various aspects of networking need to be addressed. So, we proposed a novel architecture called multi-agent-based network automation of the network management system (MANA-NMS). The architecture rely on network function *atomization*, which defines *atomic* decision-making units. Such units could represent VNFs. These *atomic* units are autonomous and adaptive. In this part, first, we present a theoretical discussion of the challenges arisen by automating the decision-making process. Next, the proposed multi-agent system is presented along with its mathematical modeling. And then MANA-NMS architecture is mathematically evaluated from functionality, reliability, latency, and resource consumption performance perspectives. As an example for *atomic* agent design, we have developed an autonomous network traffic classifier agent (NTCA). We design and implement an NTCA using a machine learning algorithm as a cognitive component of the agent. To compare, we used K-Nearest Neighbors (K-NN), Decision Tree, Support Vector Machine (SVM), and Naive Bayes in the agent design. We perform an evaluation using classification accuracy, training latency, and classification latency. We also tested the performance of the NTCA by implementing it in the MANA-NMS conceptual framework.

The third part of this P.h.D. work is to use MANA-NMS principles to decomposition SDN controllers and other monolithic systems and incorporate intelligence in the subfunctions, creating loosely coupled units in the service-oriented architecture. The existing controllers are monolithic, resulting in code inefficiency for distributed deployment. *microONOS* controller has been proposed, showing a decomposed controller architecture into logical subfunctions. These functions are implemented as microservices and deployed as VNF, enabling flexible deployment. However, the microONOS controller is in early-stage development and the full controller decomposition is not availed. Moreover, the communication interface between the decomposed components of the controller is based on gRPC. Our proposed architecture implements Ryu controller decomposition. In the decomposition, we used REST API as a communication interface between the decomposed functions. Moreover, we compared the performance using gRPC and WebSocket. We also further proposed a multi-agent architecture for the next-generation network. In this regard, recently, the 3GPP standard defines the service-based architecture (SBA) framework, where the architecture elements are defined in terms of Network Functions (NFs). This approach provides flexibility in terms of dynamic scaling and backup deployment of functions. However, to fully utilize the flexibility and dynamicity that the architecture provides, intelligence should be introduced in the decomposed functions. Here, we propose to unify well-defined standards for the 5G architecture such as Software-defined Networking (SDN), ETSI network function virtualization (NFV), ETSI generic autonomic networking architecture (GANA), and ETSI multi-access edge computing (MEC) in a unified intelligent architecture. Moreover, we define network functions and applications as atomic units, as in the case of MANA-NMS. Using these agents as building blocks, we provide an intelligent pool of networking resources and applications that can collaborate to form next-generation architectures for future 6G networks.

Acknowledgements

First of all, I would like to thank Prof. Fabrizio Granelli for giving me the wonderful opportunity to join the P.h.D school. I also appreciate his friendly supervision and tremendous support during the last three years. His philosophy of relaxed and limited constraint type of supervision has enabled me to have the freedom to think independently, developing self confidence, and grow as a mature academic researcher. Moreover, I would like to extend my gratitude to a colleague of mine Dr. Riccardo Bassoli for his meticulous review, suggestion, and comments on my work during the PhD period in preparing, and submitting articles for journals or conferences publications. I also would like to thank the University of Trento administrative stuff for their unwavering support under any difficulties, especially during the COVID pandemic period.

Beyond the academic world, my family has played great role in encouraging me to be successful. Without their support, it would have been nearly impossible to succeed as-such. In this regard, my wonderful mother has played great role and so does my uncles, especially Eng. Adane Telore in cultivating and upbringing me to this end. Their support lasted almost all my entire life. The most important moment was when I had been having a difficult time when I lost my amazing father, who I dedicate this work too. The sudden loss of him has done a huge physiological and financial damage to me and our family as he was the main provider of the family at the time. At that difficult time, as a young 14 year dreamer boy, my dreams were big which momentarily felt shattered and the doors of my unambitious future seamed to been closed.

However, my family said big NO to my fear, and standing by my side, they said your a blessing to our family, nothing will be lost. So thanks to these wonderful, courageous, and forward looking mother, uncles, ant, and grandmother, I have become who I wanted to be while fulfilling my dream. The name "Adane" means to save and the name "Mulu" means full in Amharic. These are the name of my elder uncle and my mother, respectively. Together, they saved me and made me a complete person living up to their name.

In all these, even if I sometimes doubt His existence and His involvement in our active life, without any doubt GOD has everything to me and my family. As a novice scientist, I challenged His whereabouts and how He operates. However, during this PhD period, He has worked in a mysteries way to support me in fulfilling my initial dreams. Proving me terribly wrong! THANK YOU ALMIGHTY GOD.

I would also like to thank my wonderful Ethiopian, Italian, and others friends from across the globe, who I meet in Trento, Italy and Albuquerque, USA. Trento in fact is a wonderful international city and so does the University of Trento. The combination of the two has created a tremendously vibrant and multi cultural environment, where I entertained and learned a lot of interesting life lesson along with so much fun. Even during the last difficult year, the city of Trento and the University of Trento have played very important roles in tackling the difficulties of COVID pandemic through various means, including the scientific contribution towards the solution of ending the pandemic. Thank you Trento, my second home next to my birth city Wonji!

Finally, I would like to appreciate the funding agency for my P.h.D. This work has been partially funded by NATO Science for Peace and Security (SPS) Programme in the framework of the project SPS G5428 ”Dynamic Architecture based on UAVs Monitoring for Border Security and Safety”.

Contents

Acknowledgements **ix**

1 Introduction to Network Automation and Network Softwarization **3**
- 1.1 Introduction . . . 3
- 1.2 Network Softwarization: Overview and Motivation . . . 7
  - 1.2.1 Network Softwarization . . . 7
  - 1.2.2 Software Defined Networking (SDN) . . . 8
  - 1.2.3 Network Function Virtualization . . . 9
- 1.3 Overview of Network Management Systems and Network Automation 10
  - 1.3.1 Network Management and Control as a Cyclic Process . . . 11
- 1.4 Edge Computing, Cloudification, and Containerization . . . 13
  - 1.4.1 Containerization and Cloudificaton . . . 13
  - 1.4.2 Edge Computing . . . 14
  - 1.4.3 C-RAN . . . 15
- 1.5 Microservice and Multi-Agent Systems for Autonomic Networking . . 16
  - 1.5.1 Microservice Architecture . . . 16
  - 1.5.2 Application of Multi-Agent for Network Automation . . . 17
- 1.6 Artificial Intelligence and Machine Learning for Networking . . . 18
- 1.7 LTE, 5G, B5G, and 6G . . . 20
  - 1.7.1 LTE Technologies . . . 20
  - 1.7.2 5G Technologies . . . 20
  - 1.7.3 5G Service Requirements . . . 22
  - 1.7.4 B5G, and 6G . . . 23

2 Mathematical Models for Softwarized Networks, VNF Design, and Softwarized LTE/5G Functions' Local Deployment **25**
- 2.1 Introduction . . . 25
- 2.2 Existing VNF Placement Algorithms . . . 26
  - 2.2.1 Resource Optimization . . . 26
  - 2.2.2 Minimizing Service Delay . . . 26
  - 2.2.3 Minimizing Energy Consumption . . . 26
  - 2.2.4 Maximizing Reliability, Availability and Service Admission . . 27
  - 2.2.5 VNF Placement in Cloud, Fog and Edge Data Center Computing 27
  - 2.2.6 Load balancing and Traffic Steering in Service Function Chain . 27
- 2.3 C-RAN Modeling as a Multi-layer Loopless-random Hypergraph . . . 28
  - 2.3.1 System Model . . . 28
  - 2.3.2 Service Modeling . . . 30
  - 2.3.3 System Model Formulation . . . 31
    - End-to-end Service Latency . . . 32
    - End-to-end Service Reliability . . . 34
    - Edge Data Center's Computing Capacity Constraint . . . 34
    - End-to-End Power Consumption . . . 35
    - Throughput Constraint . . . 36

Overall Service Admission Probability 38
Overall Multi-Objective Formulation 39
2.3.4 System Model Formulation with Service Differentiation 42
Improving VNFs Reliability by Using Backup VNFs for Critical Services 42
Allocating More Processing Resources to Reduce Edge Data Center Processing Latency for Critical Services 42
Energy Efficiency Calculation 43
2.4 Performance Evaluation of the VNF Placement Mathematical Model . 43
2.4.1 Reliability and Latency Constraint Analysis 45
2.4.2 Throughput Constraint Analysis 46
Edge Data Center Computational Capacity Constraint Analysis 48
2.4.3 Power Consumption and Energy Efficiency Analysis 52
2.5 A Translator as Virtual Network Function for Network Level Interoperability of Different IoT Technologies 53
2.5.1 Overview of IoT Technology Interoperability 55
2.5.2 Proposed Network Level Interoperability Architecture 58
2.5.3 A VNF Based IoT Interoperability Translator Simulation and Performance Evaluation 61
2.6 Emulation of LTE, LTE-A, and 5G Over a Lightweight Open-Platform: Re-configuration Delay Analysis 65
2.6.1 Overview of Mobile and Cloud Technologies 66
LTE Technologies 66
5G Technologies 67
2.6.2 Cloud Based Wireless Technologies Deployment 68
Kubernetes 68
2.6.3 4G/5G Emulation Testbed and Performance Evaluation 70
Creation of Kubernetes Cluster Using K3d 70
Migration of 4G/5G docker container images to Kubernetes cluster 73

3 Multi-Agent Based Network Management Automation 77
3.1 Introduction 77
3.2 State of the Art in Multi-Agent Based Autonomic Networking 80
3.2.1 Autonomic Networking: Historical Perspective 80
Autonomic Networking before the Introduction of SDN and NFV 80
Autonomic Networking After the Widespread Adoption of SDN and NFV 81
3.2.2 Multi-Agent Systems for Autonomic Networking 83
3.3 Network Management Automation Challenges 84
3.3.1 Decision Theory from Network Automation Perspectives . . . 85
Data Analytics and Cognition 87
Generating Cumulative Decision 87
Decision Organization 88
Overall Autonomic Decision Assessment and Verification . . . 88
Decision Execution 89
Monitoring System Behavior 89
3.4 Multi-Agent Based Autonomic Network Management System 89
3.4.1 Multi-Agent based Network Management System 90

3.4.2 Multi-Agents and Microservices for Decentralized and Loosely-coupled Softwarized Systems 91
3.5 Proposed MANA-NMS Architecture 91
3.5.1 Network Functions *Atomization* 92
3.5.2 Proposed MANA-NMS Architecture in Comparison with GANA Architecture 93
Proposed MANA-NMS Architecture in Comparison with SDN-NFV Architecture 94
3.6 Mathematical Representation of Multi-agent Autonomic Network Management System 96
3.6.1 Service Arrival, Scheduling, and Admission 99
3.6.2 Service Latency in Edge Data Center 100
3.6.3 Data Center Based Multi-Agent System Reliability 101
3.7 MANA-NMS Architecture's Performance Evaluation 102
3.7.1 System Description and Simulation Environment Specification 102
3.7.2 Service Arrival and Processing Evaluation 107
3.7.3 Agent Utilization and Edge Data Center Resource Consumption 110
3.7.4 Service Queuing and Processing Latency Evaluation 113
3.7.5 System Evaluation for Fail-over Scenarios 115
3.7.6 System Reliability Evaluation and Service Differentiation for Resource Efficiency 116
3.8 Autonomous Network Traffic Classifier Agent for Autonomic Network Management System 119
3.8.1 Overview and Background 119
3.8.2 Machine Learning Models For Network Traffic Classification . 120
Decision Tree Classifier Algorithm 121
Naive Bayes Classifier Algorithm 121
Support Vector Machine Classifier Algorithm 122
K-Nearest Neighbours Classifier Algorithm 123
3.8.3 Proposed Network Traffic Classifier Agent 124
Agents 124
Network Traffic Classifier Agent Architecture 124
Communication in MANA-NMS 125
3.8.4 Performance Evaluation of Classifier Agent 125
Conceptual Framework for the Network Traffic Classifier Agent Implementation 126
Evaluation Results 127

4 SDN Controller Decomposition Using Microservice 131
4.1 Introduction 131
4.2 Background and Motivation 133
Background 133
4.3 Overview of SDN Controller Components 133
4.3.1 Ryu SDN Framework 135
4.4 Motivation for SDN Controller Decomposition 138
4.5 Microservices-Based SDN Controller Decomposition 139
4.5.1 Decomposing SDN Controller 142
4.5.2 SDN Controller Internal Components as a Microservice 142
4.5.3 External Applications as a Microservice 143
4.5.4 Communication Interface Between Decomposed Services 144

Communication Interface Between the decomposed Microservice based Controller Functions 144
4.6 Decomposed SDN Controller TestBed Implementation and Performance Results 145
4.6.1 Decomposing Ryu SDN Controller 146
4.6.2 Experimental Environment 147
4.6.3 Benchmark of Network Communication Protocols 148
4.6.4 Resilience and Scalability Test 149
Resilience Test 150
Scalability Test 150

5 MANA-NMS based Unified Architecture and Its Application in 6G and Next Generation Network Management Systems 155
5.1 Introduction 155
5.1.1 Motivation and Background 156
5.1.2 Microservices-based SDN Controller 157
5.1.3 ETSI SDN-NFV MANO 157
5.1.4 ETSI GANA 158
Generic Autonomic Network Architecture 158
Multi-Agent System 158
5.1.5 3GPP-ETSI MEC Architecture 159
5.1.6 Proposed Unified Grad Architecture 159
5.1.7 Proposed Unified Architecture 159
5.1.8 Network Function Atomization 160
5.1.9 Agent Internal Architecture 161
5.1.10 Multi-Agent Based SDN Controller Decomposition 161

6 Future Application of MANA-NMS 163
6.1 MANA-NMS for IoT Connectivity Technology Management for Space Application 163
6.1.1 Internet of Things 163
6.1.2 Network Coverage, Network Softwarization, and Network Automation 165
Backbone Network Technologies for Space Applications 165
Network Coverage Technologies on Remote Environment . . . 166
Virtualization and Softwarization 167
Network Automation 167
6.1.3 Artificial Intelligence for Space Applications 168
6.1.4 Cloud, Edge, and Fog Computing 168
Teleoperation Using Edge Computing 169
6.2 Summary and Conclusion 170

Bibliography 173

List of Figures

1.1 SDN architecture. 6
1.2 High-level NFV framework [189]. 6
1.3 Automation Enabling Technologies. 7
1.4 Unified SDN-NFV Architectural Framework. 8
1.5 SDN Management Architecture with respect to autonomic management[97] 9
1.6 ETSI-NFV Architectural Framework[189] 10
1.7 Cyclic process of network management. 11
1.8 Cloud computing Vs. Fog computing Vs. Edge computing. 15
1.9 Downlink communication in heterogeneous 4G or LTE RAN and heterogeneous 5G Cloud RAN. The latter places baseband processing at virtual baseband units (BBUs) in operators' data centers and run them as virtual machines or virtual functions in containers. RRH, radio remote head 16
1.10 Microservice Architecture 17
1.11 4G/5G Core Network 21
1.12 enhanced mobile broadband (eMBB), ultra-reliable low-latency communications (URLLC), and massive machine-type communications (mMTC) 22

2.1 4G Architecture, 5G with Cloud Architecture and 5G with Edge Data Center Architecture 26
2.2 Physical Infrastructure and Representation of 5G Cellular Network . . 29
2.3 RAN, Backhaul and Edge Data Center 36
2.4 Sample C-RAN Infrastructure Deployed for Performance Evaluation Scenario 44
2.5 Total Service Throughput Demand 46
2.6 Admission and Rejection of Service Throughput Due to RAN Constraint 47
2.7 Admission and Rejection of Service Throughput at Microwave Backhaul 47
2.8 Admission and Rejection of Services Throughput at Fiber Backhaul . . 48
2.9 Admission and Rejection of Services Throughput at the Edge Data Center Network 1 Tbps Aggregation Links 48
2.10 Arriving Workload Traffic Distribution 49
2.11 Workload Requested and Rejected 49
2.12 Total Workload Requested and Rejected using Backup VNFs and CPU Over-provisioning for Reliability and Latency Improvements, Respectively 50
2.13 Total Workload Requested and Rejected using Backup VNFs and CPU Over-provisioning for Reliability and Latency Improvements Respectively with Service Differentiation 51
2.14 Total Workload Admitted 51
2.15 Edge Data Center Workload Admission Probability 51

2.16 Used CPU Resources Comparison for the Given Service Workload. . . . 52
2.17 Total Power Consumption of Edge Data Center in Different Cases . . . 52
2.18 Energy Efficiency of Edge Data Center 53
2.19 Overall performance of the proposed model in comparison with the existing 4G performance and the expected 5G requirements 54
2.20 Proposed Virtualized Solution Structure. 59
2.21 Proposed Virtualized Solution Workflow 60
2.22 Implemented Network Topology 62
2.23 Translation Delay 63
2.24 Translation and Transmission Delay 63
2.25 Transmission Delay at Various Traffic Load Conditions 64
2.26 Transmission Delay Using a Virtualized Translator at Various Traffic Load Conditions 64
2.27 4G/5G Core Network 67
2.28 Kubernetes service for both open5gs EPC and freee5gc 5GC 70
2.29 Final architecture for GEPC/5GC switch using Kubernetes service . . 71
2.30 Components and their interconnection for 4G EPC 71
2.31 Components and their interconnection for 5GC 72
2.32 Comparison between kind and k3d 74
2.33 CPU usage of CN pod 75
2.34 Memory usage of CN pod 75
2.35 CPU usage of eNB pod 75
2.36 Memory usage of eNB pod 75
2.37 Overall CPU usage of kubernetes local cluster 76
2.38 Overall memory usage of kubernetes local cluster 76

3.1 Unified SDN-NFV Architectural Framework [148] 79
3.2 Network Management Cycle 85
3.3 Multi-agent Based Representation of MANA-NMS for Cloud RAN . . 93
3.4 Simple Example for Network Level Agent Relationship 94
3.5 Multi-Agent Based Representation of GANA Architecture. 95
3.6 Multi-Agent-Based Network Management Automation on Unified SDN-NFV Architectural Framework. 95
3.7 Fundamental Components of Autonomic Agent 97
3.8 Agent States 98
3.9 Cyclic Step of Service Scheduling, Processing and Monitoring 103
3.10 Agent Sequencing and Scheduling for Arrival Service Processing . . . 104
3.11 Considered Network Topology For Path Computation 105
3.12 Workload Distribution 108
3.13 Average Arriving Service Workload 108
3.14 Service Rejection 109
3.15 Service Admission Probability 109
3.16 Average Number of Required Service Agents 110
3.17 Average Number of Idle Agents 111
3.18 Average Number of Busy Agents 111
3.19 Agents Utilization 112
3.20 Edge Data Center Resource Utilization 112
3.21 Agent Busy Period 113
3.22 Agents Average Communication Frequency for Database Update due to Service Arrival 113
3.23 Arrival Service Queuing Delay 114

3.24 Service Processing Delay in the Service Processing Agents 114
3.25 End-to-End Service Latency in Edge Data Center 115
3.26 Impact of Server (Hosting Service Processing Agents) Failure on Service Rejection 116
3.27 Impact of Server Failure on End-to-End Latency in the Edge Data Center 117
3.28 Impact of Server Failure on Service Admission Probability 117
3.29 Service Rejection with Backup Service Processing Agents 118
3.30 Service Admission Probability 118
3.31 CPU Resource Consumption with Backup System 119
3.32 CPU Consumption with Backup System for uMTC Service Processing Agents 119
3.33 General overview of a MAS. 120
3.34 An Example of a Decision Tree. 121
3.35 An Illustration of a Hyperplane and Support Vectors. 122
3.36 An Illustration of the K-NN Classifier Algorithm in action. 123
3.37 Network Traffic Classifier Agent Architecture 125
3.38 The Process of Traffic Classification. 126
3.39 Proposed MAS Model. 127
3.40 Comparison of the Training Latency of the Network Traffic Classifier Agent Designs. 129
3.41 Comparison of the Classification Latency of the Network Traffic Classifier Agent Designs. 130

4.1 A Unified Architecture for SDN and NFV 134
4.2 ONF's Software-Defined Network Architecture 135
4.3 Ryu Internal Monolithic Logical Architecture 136
4.4 Microservices-based SDN deployment blueprint in IIoT Scenarios . . . 140
4.5 Architectural Overall of Proposed Microservices-based SDN Controller Decomposition 142
4.6 Proposed microservices-based SDN controller sample deployment architecture 144
4.7 Ryu-based MSN Implementation Architecture 146
4.8 Mininet Topology for Experimental Testbed 148
4.9 Performance results 149
4.10 Resilience Test: Average Message Delay 150
4.11 Scalability test: scenario 1 (multiple NS) 151
4.12 Scalability test: Scenario 2 (Single NS - multiple VNF) 152
4.13 Scalability test: Network Topology 152
4.14 Scalability test: Average Latency for the First Packet 153
4.15 Scalability test: Average Latency for Normal Flow 153

5.1 Architectures of SDN decomposed control plane (top left), ETSI SDN-NFV MANO (top right), ETSI GANA (bottom left), and 3GPP-ETSI MEC (bottom right). 156
5.2 Microservices and Multi-Agent Based Unified Architectural Model for ETSI GANA, ETSI MEC, ETSI NFV, and SDN 160
5.3 Agent Internal Architecture 161
5.4 Agent Instantation From a Template 162

List of Tables

2.1 The three type of services and their requirements 45
2.2 Different Components of C-RAN and Their Constraints 45
2.3 Values for emulation components docker image size 74

3.1 Microservice and Multi-Agent Comparison 91
3.2 Internal-Functional Specification of Agents 105
3.3 Detailed Agents Parametric-Specifications Used in Developing the overall Simulation System 106
3.4 Service Specification 106
3.5 Simulation Parameters 106
3.6 Classification Accuracy of the different Network Traffic Classifier Agent Designs. 128

4.1 Comparison between SDN Monolithic architecture and Microservices-based SDN architecture [283] 141

List of Abbreviations

SDN	**S**oftware **D**fiend **N**etworking
NFV	**N**etwork **F**unction **V**irtualization
IoT	**I**nternet **o**f **T**hings
C-RAN	**C**loud **R**adio **A**ccess **N**etwork
RAN	**R**adio **A**ccess **N**etwork
RRH	**R**adio **R**emote **H**heads
ML	**M**achine **L**earning
AI	**A**rtificial **I**ntelligence
GANA	**G**eneric **A**utonomic **N**etwork **A**rchitecture
ANMS	**A**utonomic **N**etwork **M**anagement **S**ystem
ANM	**A**utonomic **N**etwork **M**anagement
CAPEX	**Cap**ital **Ex**penditure
OPEX	**Op**erational **Ex**penditure
BBU	**B**ase **B**and **U**nit
S-GW	**S**erving **G**ateway
P-GW	**P**acket **G**ateway
VNF	**V**irtual **N**etwork **F**unction
M2M	**M**achine **T**o **M**achine
H2T	**H**uman **To** **T**hing
T2T	**T**hing **To** **T**hing
IBM	**I**nternational **B**usiness **M**achines
ACL	**A**utonomic **C**ontrol **L**oop
DoS	**D**enial **o**f **S**ervice
NFVI	**N**etwork **F**unction **V**irtualization **I**nfrastructure
NAT	**N**etwork **A**ddress **T**ranslation
MANO	**M**anagement **a**nd **N**etwork **O**rchestration
QoS	**Quality** **o**f **S**ervice
SBI	**S**outh **B**ound **I**nterface
NBI	**N**orth **B**ound **I**nterface
SDNVA	**SDN** based **N**etwork **V**irtualization **A**rchitecture with Autonomic Management
MAPE-K	**M**onitor **A**nalyze **P**lan **E**xecute **K**nowledge
VM	**V**irtual **M**achines
eMBB	**e**nhanced **M**obile **B**road**B**and
URLLC	**U**ltra-**R**eliable **L**ow-**L**atency **C**ommunications
mMTC	**m**assive **M**achine-**T**ype **C**ommunications
VIM	**V**irtual **I**nfrastructure **M**anager
4G	**F**ourth-**G**eneration
LTE	**L**ong-**T**erm **E**volution
NMS	**N**etwork **M**anagement **S**ystems
μONOS	**micro** **O**pen **N**etwork **O**perating **S**ystem
OS	**O**perating **S**ystem
CPU	**C**entral **P**rocessing **U**nit
MANA-NMS	**M**ulti **A**gent based **N**etwork **A**utomation of the **N**etwork **M**anagement **S**ystem

B5G	**B**eyond **5G**
REST	**R**epresentational **S**tate **T**ransfer
gRPC	**g**RPC **R**emote **P**rocedure **C**alls
NATO	**N**orth **A**tlantic **T**reaty **O**rganization
SPS	**S**cience **P**eace and **S**ecurity
UAV	**U**nmanned **A**real **V**ehicle
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
DSP	**D**igital **S**ignal **P**rocessing
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
IFFT	**I**nverse **F**ast **F**ourier **T**ransform
SOA	**S**ervice **O**riented **A**rchitecture
CDMA	**C**ode **D**ivision **M**ultiple **A**ccess
MAS	**M**ulti **A**gent **S**ystem
WSN	**W**ireless **S**ensor **N**etwork
MDN	**M**anual **D**efined **N**etwork
IDN	**I**ntelligent **D**efined **N**etworks
z-TORCH	**z**ero **To**uch **Orch**estration
LTE-A	**L**ong **T**erm **E**volution- **A**dvanced
ETSI	**E**uropean **T**elecommunications **S**tandards **I**nstitute
MEC	**M**obile **E**dge **C**omputing
vBBU	**v**irtual **B**ase**B**and **U**nit
VIM	**V**irtual **I**nfrastructure **M**anager
SFC	**S**ervice **F**unction **C**haining
SLA	**S**ervice **L**evel **A**greement
PP	Random **P**oint **P**rocesses
PPP	**P**oisson **P**oint **P**rocesses
MW	**M**ultiplicatively **W**eighted
IIoT	**I**ndustrial **I**nternet **o**f **T**hings
MW	**M**micro**w**ave
MIMO	**M**ultiple **I**nput **M**ultiple **O**utput
ITU	**I**nternational **T**elecommunication **U**nion
IMT	**I**nternational **M**obile **T**elecommunications
BS	**B**ase **S**tation
GOPS	**G**iga **O**peration **P**er **S**econd
PM	**P**hysical **M**achine
MTBF	**M**ean **T**ime **B**etween **F**ailure
MTTR	**M**ean **T**ime **T**o **R**epair
MILP	**M**ixed **I**nteger **L**inear **P**rogram
MIP	**M**ixed **I**nteger **P**rogram
NBIoT	**N**arrow**B**and **IoT**
LoRa	**Lo**ng **Ra**nge
VAP	**V**irtualized **A**ccess **P**oint
WoT	**W**eb **o**f **T**hings
XML	Full E**x**tensible **M**arkup **L**anguage
JSON	**J**ava**S**cript **O**bject **N**otation
WSDL	**W**eb **S**ervices **D**escription **L**anguage
REST API	**R**epresentational **S**tate **T**ransfer **A**plication **P**rogramming **I**nterface
SDR	**S**oftware **D**efind **R**adio
CoAP	**Co**nstrained **A**pplication **P**rotocol
MQTT	**M**essage **Q**ueue **T**elemetry **T**ransport
BLE	**B**luetooth **L**ow **E**nergy

IP	**I**nternet **P**rotocol
IPv6	**I**nternet **P**rotocol **v**ersion **6**
6LoWPAN	IPv**6** over **Lo**w Power **W**ireless **P**ersonal **A**rea **N**etworks
SQL	**S**tructured **Q**uery **L**anguage
HTTP	**H**yperText **T**ransfer **P**rotocol
Wi-Fi	**Wi**reless **Fi**delity
AP	**A**ccess **P**oint
IaaS	**I**nfrastructure **a**s **a** **S**ervice
3GPP	**3rd** **G**eneration **P**artnership **P**roject
6G	**6** **G**eneration Network
ITU	The **I**nternational **T**elecommunication **U**nion
ITU-T	**T**elecommunication **S**tandardization **S**ector
ONF	**O**pen **N**etwork **F**oundation
DE	**D**ecision **E**etwork **E**lements

This work is dedicated to my
wonderfully charming father
Tadessse Arzo, my humble
grand father Telore Tissore, and
my vibrant uncle Tesfalidet
Telore that are tragically passed
away and are being missed from
our lovely, laughing, and
hardworking family

Authors List Of Publications

1. **S. T. Arzo**, D. Scotece, R. Bassoli, D. Barattini, F. Granelli, L. Foschini, and F. H. P. Fitzek, “MSN: a playground framework for desing and evaluation of MicroServices-based sdN controller” J Netw Syst Manage 30, 19 (2022). https://doi.org/10.1007/s10922-021-09631-7

2. C. Naiga, **S. T. Arzo**, F. Granelli, R. Bassoli, and F. H. P. Fitzek, “Autonomous Network Traffic Classifier Agent for Autonomic Network Management System”, Accepted to IEEE Globecom 2021

3. N. K. Ostinelli, **S. T. Arzo**, F. Granelli, and M. Devetsikiotis, “Emulation of LTE/5G Over a Lightweight Open-Platform: Re-configuration Delay Analysis”, Accepted to IEEE Globecom 2021

4. **S. T. Arzo**, F. Zambotto, F. Granelli, R. Bassoli, M. Devetsikiotis and F. H. P. Fitzek, “A Translator as Virtual Network Function for Network Level Interoperability of Different IoT Technologies”,2021 IEEE 7th International Conference on Network Softwarization (NetSoft), 2021, pp. 416-422, doi: 10.1109/NetSoft51509.2021.9492677.

5. **S. T. Arzo**, C. Naiga, F. Granelli, R. Bassoli, M. Devetsikiotis and F. H. P. Fitzek, "A Theoretical Discussion and Survey of Network Automation for IoT: Challenges and Opportunity," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2021.3075901.

6. **S. T. Arzo**, R. Bassoli, F. Granelli and F. H. P. Fitzek, "Multi-Agent Based Autonomic Network Management Architecture," in IEEE Transactions on Network and Service Management, doi: 10.1109/TNSM.2021.3059752.

7. **S. T. Arzo**, R. Bassoli, F. Granelli and F. H. P. Fitzek, "Study of Virtual Network Function Placement in 5G Cloud Radio Access Network," in IEEE Transactions on Network and Service Management, vol. 17, no. 4, pp. 2242-2259, Dec. 2020, doi: 10.1109/TNSM.2020.3020390.

8. R. Bassoli, F. Granelli, **S. T. Arzo**, and M. D. Renzo, “Toward 5G cloud radio access network: An energy and latency perspective,”Trans. Emerg.Telecommun. Technol., vol. 27, no. 1, pp. 433–446, May 2019.

9. DZIMITRY KLIAZOVICH, **SISAY T. ARZO**, AND OTHERS, “ACCOUNTING FOR LOAD VARIATION IN ENERGY-EFFICIENT DATA CENTRE”, IEEE ICC 2013 CQRM SYMPOSIUM PUBLISHED, JUN 2013.

10. DZIMITRY KLIAZOVICH, **SISAY T. ARZO**, AND OTHERS, “ENERGY-EFFICIENT SCHEDULING FOR CLOUD COMPUTING APPLICATIONS WITH TRAFFIC LOAD BALANCING”, IEEE GREENCOM 2013 SYMPOSIUM, BEIJING, CHINA.

11. **S. T. Arzo**, D. Scotece, R. Bassoli, F. Granelli, L. Foschini, and F. H. P. Fitzek, “A 6G Agent Based Intelligent Network Architecture”, Submitted for IEEE Communication Standard Magazine

12. **S. T. Arzo**, D. Sikeridis, M. Devetsikiotis, F. Granelli, R. Fierro, "A Survey of Essential Technologies and Concepts for Massive Space Exploration: Challenges and Opportunities", submitted to IEEE Transactions on Aerospace and Electronic Systems.

13. **S. T. Arzo**, C. Naiga, F. Granelli, R. Bassoli, M. Devetsikiotis and F. H. P. Fitzek, "Multi-Agent Based Traffic Prediction and Traffic Classification for Autonomic Network Management Systems for 6G Networks" submitted to Special Issue on “Edge Intelligence for 6G Networks”, Computer Communications, Elsevier.

14. H. Haridy, **S. T. Arzo**, F. Granelli, R. Bassoli, and F. H. P. Fitzek, “Agent Based Autonomous Network Traffic Prediction: Design and Analysis”, To be submitted

Chapter 1

Introduction to Network Automation and Network Softwarization

1.1 Introduction

Legacy communication networks were comprised of several devices such as routers, switches, servers, firewalls, and end-user devices. These reasonably small-sized networks were managed mainly by human supervision [159, 233]. A human administrator was in change of configuring and making the desired changes in the settings of the network. This was possible for the small-sized networks.

However, over the past decades, communication technologies and networks have tremendously evolved enabling the connection of a large number of devices and applications for billions of users. Such networks have a huge capital expenditure (CAPEX) since new devices and services are implemented. This results in complex networks that are difficult to manage. It has become humanly infeasible to manage such networks as they grow thereby increasing the operational expenditure (OPEX) since more human resources and expertise are needed.

The need for network automation is more pronounced when we consider the requirements of a future network such as internet of things (IoT). IoT is the interconnection of highly heterogeneous networked entities and the networks that follow several different communication patterns, such as machine to machine (M2M), human-to-human (H2H), human-to-thing (H2T), thing-to-thing (T2T), or thing-to-things (T2Ts). With newer wireless networks, superior sensors, self-healing soft-electronics, and revolutionary computing capabilities, the IoT is the frontier in the race for technological advancement. IoT also has a huge potential to revolutionize space exploration. In all of these applications, network automation plays an important role.

In general, in the past decades, communication systems have experienced several technological advancements that have facilitated an increase in the use of the Internet and the number of devices [187]. As a result of this development, networks are evolving, showing an increase in network size and complexity. In parallel, networks' hardware and computational complexity are also augmenting. However, this makes network management very expensive and humanly infeasible. As networks grows becoming very complex, there is a need for more efficient network management methods. The challenges of managing complex networks are numerous.

Autonomic networking is management solution that equips the network system with self-X properties; X can represent managing, configuring, healing, and protecting capability of the network [187]. Networks designed with self-management capabilities are able to independently predict, diagnose and circumvent problems with the network functions. In this sense, an autonomic network management system (ANMS) is able to capture, respond and adapt to the dynamic and evolving behavior of the network, according to the users' and services' demands. With ANMS, networks will be equipped with self-managing capabilities to make decisions both in a supervised and unsupervised manner [278]. This ensures simplified control and management of the networks and their associated services.

Autonomic networking is required to scale up the network management capability to address the expected big growth in the next-generation networks. Moreover, different stringent performance indicators should concurrently be satisfied such as latency, reliability, and service continuity. This also requires increasing expenditure and inefficiency as the network size expands. In other words configuration, monitoring, control, and management of such networks become complex and costly [239]. This means, from a business perspective, network operators need to increase revenues, while reducing CapEx and OpEx.

Recently, with the introduction of the concept of network softwarization, the capabilities of network management are enhanced by the programmability and adaptability of the networking infrastructure. Software define networking (SDN) and network function virtualization (NFV) have become de-facto standards in *network softwarization*. It is for several reasons such as network flexibility and innovation, which could lead to reduced CapEx and OpEx. In particular, network orchestration is required for virtual network function (VNF) placement and service function chaining to properly steer traffic. The paradigm NFV deals with the softwarization of network functions, which are traditionally implemented in hardware, such as firewalls, serving gateway (S-GW), packet gateway (P-GW) and baseband unit (BBU). While modern networks provide a high level of programmability, still automation in network management is supported by limited intelligence in the network.

Moreover, a dramatic increase in the amount of network information will come from network softwarization and additional new verticals such as ultra-reliable massive machine-type communication. Due to this, a completely new mechanism for network management is required. Therefore, future networks [63] should mainly target alleviating the burden through autonomic networking.

Originally, network automation hails from a manifesto by International Business Machines Corporation (IBM) in 2001 [29], [37], [159], which was applying self-management on computing. Through network automation, the network can adapt to any change in its environment, self-configure, and self-optimize without any human intervention [29, 278].

In particular, this can be through an autonomic control loop (ACL) that could be implemented to collect detailed information about the network. Network data, along with data management, and users' metadata, could be extracted and used to manage the network. Collecting such network data from big networks is very challenging but critically useful to provide dynamic and adaptive responses to the evolving service demands. Networks with these properties take their actions based on a set of predefined policies. This can be overridden by the human network administration if needed.

With a typical automation method such as monitoring, analyzing, planning, and executing methods, the network can self-adapt to any changes in its environment. The following self-management properties [29] are described in IBM's manifesto:

- Self-configuring is the capability of a network to configure and re-configure itself based on predefined policies to achieve a given performance. This should happen seamlessly and with no human intervention.

- Self-optimizing means to ensure that the network always uses the available resources to provide the best possible performance even in highly-varying environments. The network should always measure its current performance and set strategies to efficiently perform in case of any deviation from the set of expectations and predefined-ideal standards.

- Self-protecting is related to the security of the network and it is a very crucial issue. Self-protection ensures that the network is able to shield itself against any potential attacks such as Denial-of-Service(DoS) attacks.

- Self-healing is needed in case of failure of any network element. A network with self-healing capabilities is able to recover from such failures in the shortest time possible. The network is able to discover and automatically repair any failed elements to ensure service continuity.

As indicated above, current networking research mostly focuses on network softwarization, relying on SDN and NFV. Softwarization maps hardware-based network functions into software. It is an important step in the automation process. SDN involves the separation of the control plane from the forwarding (data) plane [152, 97]. The forwarding plane is made up of stateless forwarding tables, that are periodically populated by the centralized controller.

On the other hand, NFV is a software implementation of network function, which is traditionally implemented in preparatory hardware[189]. The architecture of NFV contains three main components.

- Network function virtualization infrastructure (NFVI) consists of the hardware and software that host different virtual network functions (VNFs).

- VNFs are softwarized network functions such as firewall, network address translation (NAT), packet/serving-gateway(P/S-G), and baseband unit (BBU), which could be deployed in an NFVI environment.

- NFV management and network orchestration (MANO) is the place where management and orchestration of VNFs are implemented.

SDN and NFV enable the traditional static network to be flexible, paving the way for network innovation. This opens the door for rapid network evolution, transition to intelligent networking, and network automation. Today, the success of the network automation industry is motivated by SDN, NFV, and Machine Learning (ML). Even if SDN and NFV enable the possibility of network programming, virtualization, and orchestration, they do not automate network management. Therefore, a new framework should be developed to incorporate SDN for network control and programmability, NFV for virtualization and orchestration, and lastly, ML for knowledge management and cognitive ability. Figure 1.3 depicts a combination of SDN, NFV and ML for network automation.

ML has recently seen great advancements. It is expected to play a significant role in the automation process[43]. In fact, ML has been applied in various areas of networking such as traffic prediction, resource management, Quality of Service (Qo. S), and network security. ML provides cognition and reasoning in automated decision-making. The goal of ML in networks is to extract knowledge from network behavior, service, and users' behavior, using historic data for training and learning.

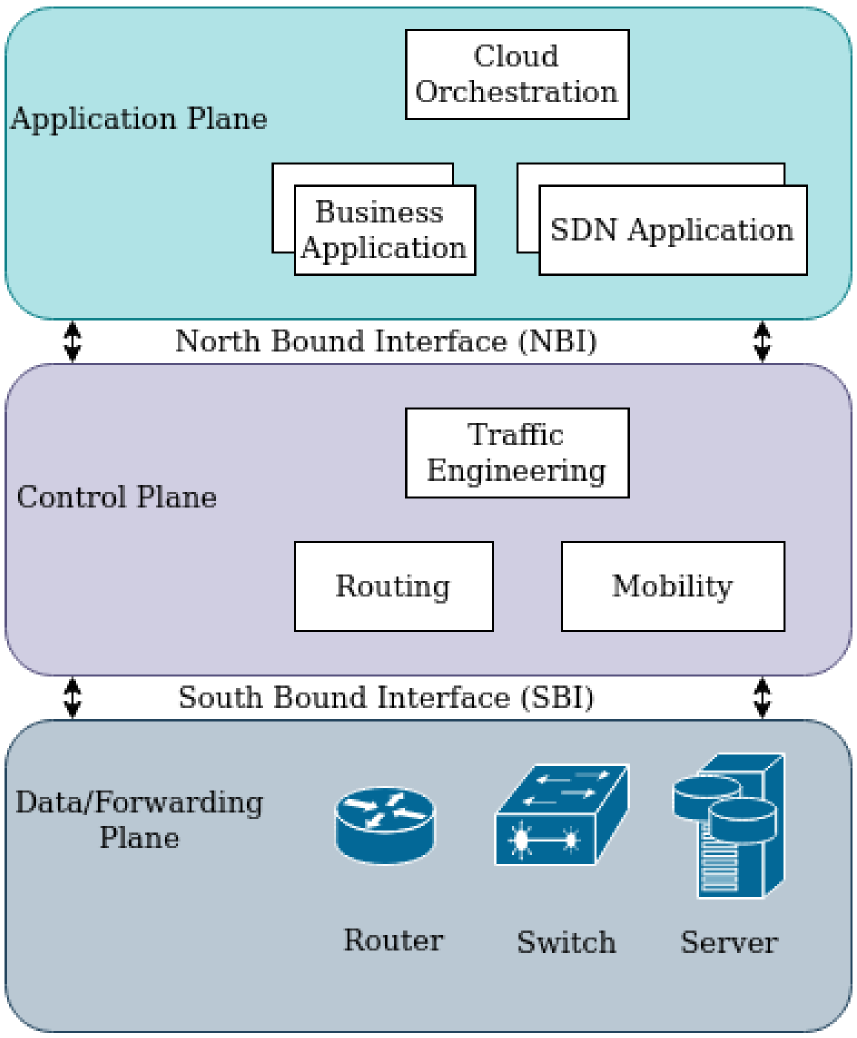


FIGURE 1.1: SDN architecture.

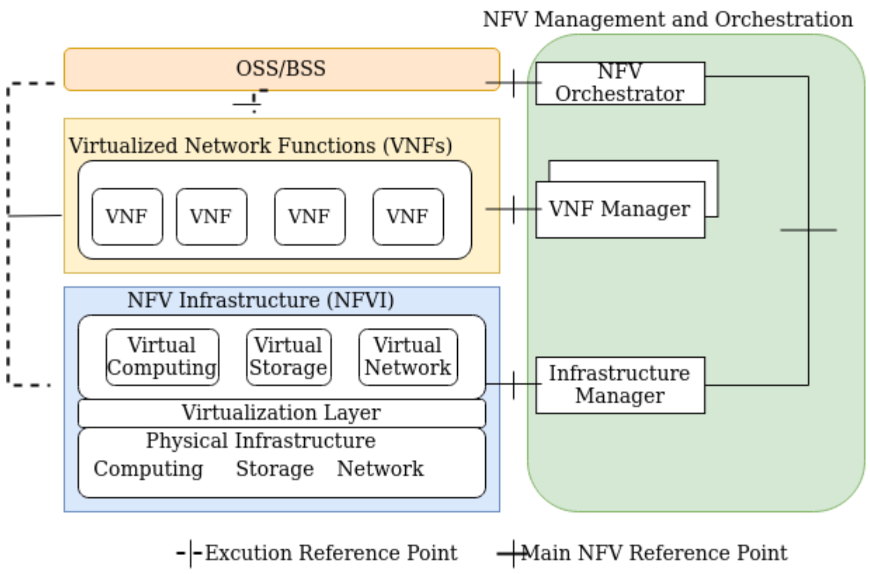


FIGURE 1.2: High-level NFV framework [189].

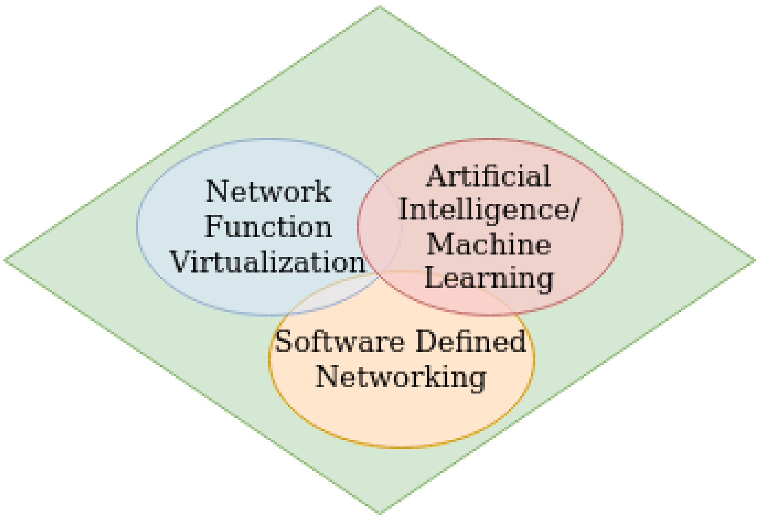

Network Function Virtualization
Artificial Intelligence/ Machine Learning
Software Defined Networking

FIGURE 1.3: Automation Enabling Technologies.

1.2 Network Softwarization: Overview and Motivation

As discussed in the introductory section, numerous research areas of networking are leading to network automation. These are network softwarization through SDN and NFV, and ML. This chapter provides an overview of network automation enabling technologies.

1.2.1 Network Softwarization

As we have introduced in the above section, SDN and NFV are the two building blocks of network softwarization [241]. They are complementary technologies. Adopting them together provides a myriad of innovative possibilities, such as dynamic network slicing, dynamic network configurations, network state measurement, and dynamic network management and control. They are enablers of network automation through the softwarization and orchestration of network functions.

By rendering the network programmable, SDN enables accelerated innovation, contributing to greater responsiveness, stability, productivity, and cost-effectiveness. Since NFV replaces devices such as load balancers, firewalls, and intrusion detection systems with software running on commodity hardware, it reduces the expense of installing the network. The forwarding, control, and management functions may also be implemented in software as in NFV [239]. These consolidated functions may then share the same resources such as computing, storage, and power. This reduces power consumption, maintenance costs and takes time to implement new services. Examples of functions that may be virtualized include switching, routing, traffic load balancing, etc. Management software that operates on converged SDN and cloud infrastructure dramatically reduce CapEx and OpEx, as well as the overall complexity of network operation [241].

Moreover, automating the network management system can supplement SDN and NFV in harnessing the full benefits of the two technologies. For instance, some of the important benefits of network slicing are network and service isolation. Enabling network slicing requires dynamic network functions orchestration and dynamic resource allocation depending on the service requirements. An automated

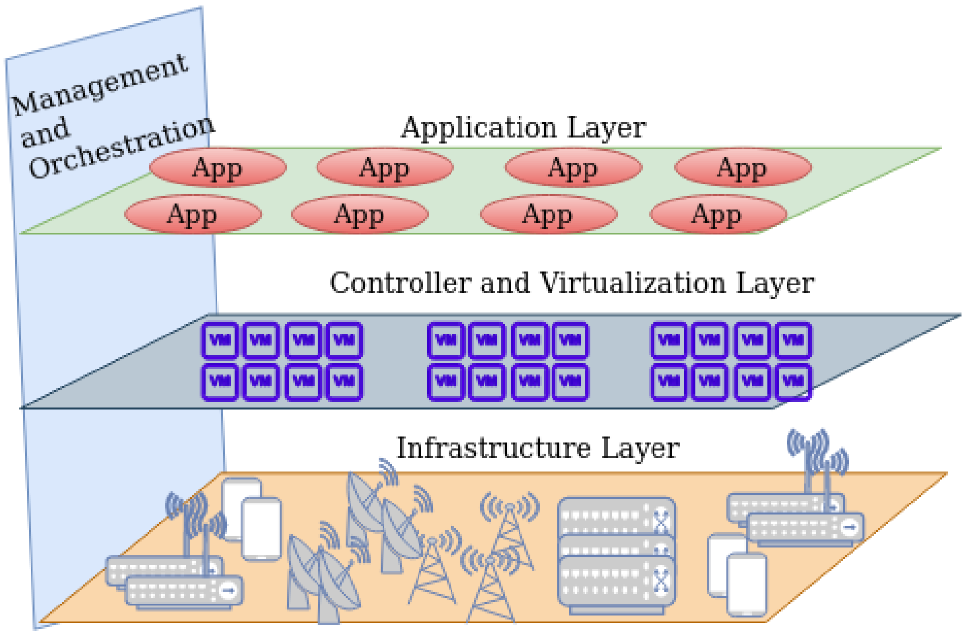

Management and Orchestration
Application Layer
App
Controller and Virtualization Layer
VM
Infrastructure Layer

FIGURE 1.4: Unified SDN-NFV Architectural Framework.

provisioning of resources and orchestration of network functions would enable satisfying the dynamic service demands. A unified architecture for SDN-NFV, which is presented in [148] is depicted in Figure 3.1.

1.2.2 Software Defined Networking (SDN)

Now let us see what SDN brings to network management. SDN is a promising solution in network automation[152, 241, 218]. This is because, SDN involves the separation of the control plane and the data plane, which significantly decreases the difficulties experienced in controlling networks. The control plane is implemented in a central controller while the hardware functions such as routing and switching can be abstracted from hardware and implemented in software [291]. This allows for a range of considerably more flexible and effective network management solutions, using network programming. This improves the role of network management in the event of any dynamic network changes. This gives room for autonomic network management (ANM). In other words, the flexibility and dynamic programmability of control enable network automation.

Figure 4.5 illustrates the SDN architecture. The data plane consists of all network devices such as switches, routers, and firewalls. The control plane and the data plane communicate over the South-Bound Interface (SBI) [38]. The concept behind SDN was to use flow tables within network devices and a common interface for configuring, controlling, and manipulating the flow table. OpenFlow is the standard protocol employed at this interface. OpenFlow protocol is used by the centralized controller to manipulate entries in flow tables. Functionalities such as fault detection, the discovery of topology changes, etc are performed by the SDN control plane [38]. The control and the application planes communicate over the north-bound interface (NBI). Over this interface, applications are able to exchange information with the control plane about the status of the network devices.

Different authors [152], [218] have proposed a number of SDN-based architectures for autonomic management. For example, [218] proposes an SDN-based network virtualization architecture with autonomic management (SDNVA) with two

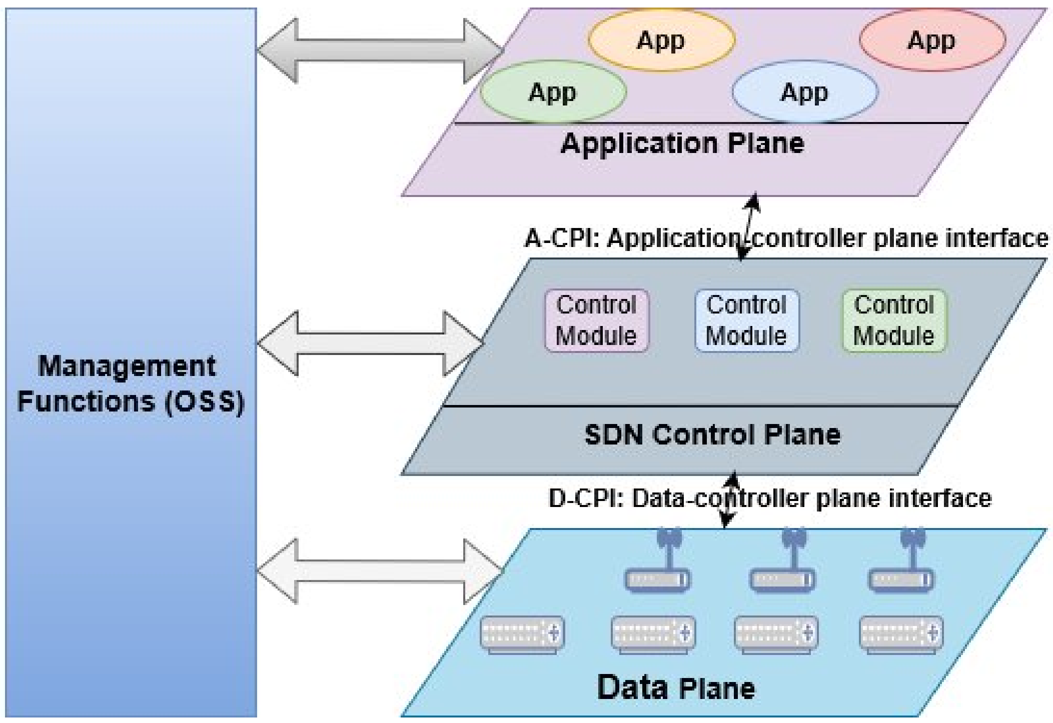

Management Functions (OSS)
App
App
App
App
Application Plane
A-CPI: Application controller plane interface
Control Module
Control Module
Control Module
SDN Control Plane
D-CPI: Data-controller plane interface
Data Plane

FIGURE 1.5: SDN Management Architecture with respect to autonomic management[97]

layers: autonomic virtual physical virtual network management. The major objective of this approach is to ensure isolation between the virtual network resources using a virtualization module and also assert a hierarchical autonomic management approach among the physical and virtual resources. Ahmed Binsahaq et. al. in [38] discussed the current state of the art literature on SDN about QoS management. The literature discussed in their work is classified based on the monitor, analyze, plan, execute, knowledge (MAPE-K) reference model [37] serve as basic functions, monitor, analyze, plan, execute, and knowledge.

1.2.3 Network Function Virtualization

NFV is another key component of network softwarization. Similar to SDN, NFV provides flexibility and programmability through dynamic orchestration of network functions. The idea of NFV is decoupling network functions from hardware and deploying them as software in virtual machines (VMs) or containers [241]. NFV originated from a white paper [56] that was presented at a conference in Darmstadt, Germany, on SDN and OpenFlow in 2012. The architecture of NFV is illustrated in Figure 1.2. It represents the three main working domains as; VNF, NFVI, and MANO [189].

A VNF is software that implements a given network function. VNFs run in one or more virtual environments(e.g. VM, containers, etc) that run on general hardware. VNFs are deployed on-demand using an NFV architecture, removing the delivery delays associated with traditional network equipment, as well as the need for on-site specialist knowledge, when deployed remotely [189]. NFVI includes physical and virtual network resources, hypervisors, VM, and virtual infrastructure managers. It also includes the physical resources for deploying and handling VNFs. The NFVI

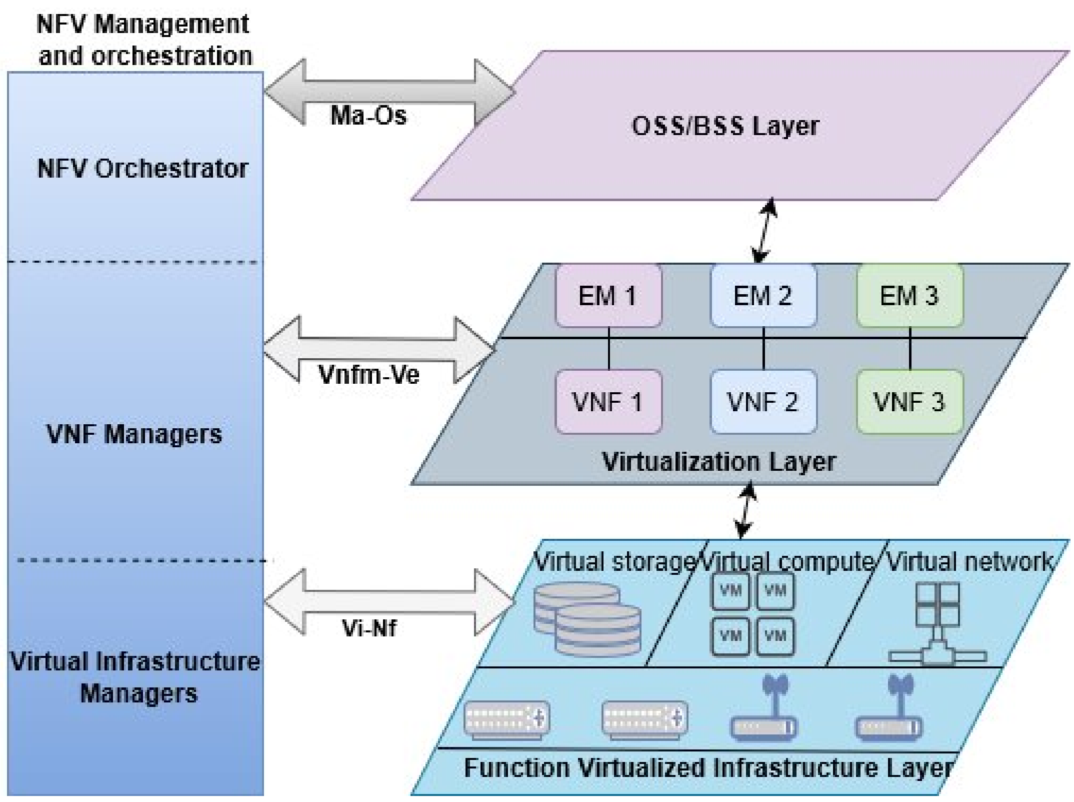


FIGURE 1.6: ETSI-NFV Architectural Framework[189]

requires the virtualization layer above the hardware, which abstracts hardware resources into virtual resources that support VNFs. The NFVI is also essential to the development of large, widely-dispersed networks without the regional limitations associated with conventional network architectures [189].

MANO is responsible for the management and orchestration of the virtualized network. NFV MANO consists of three modular components: NFV Orchestrator manages VNF enrollment, lifecycle management, global view of resource management, and NFVI resource request validation and authorization. VNF manager manages the instance management of the VNF lifecycle, provides collaboration and adaptation function for the configuration of NFVI and element/network management systems, and the recording of events. virtual infrastructure manager (VIM) controls and manages resources for the NFVI computing, storing, and networking.

1.3 Overview of Network Management Systems and Network Automation

Communication networks are very complex systems with numerous heterogeneous devices, services, and users involved. This is more pronounced in automating networks interconnecting IoT devices, as there will be a huge number of interconnected devices. For example, there are 20.6 billion connected devices in 2020[119]. Thus, managing such networks involves handling real-time events in a sophisticated and heterogeneous environment. The problem is also complicated by the continued and huge amount of data generated by IoT devices, users, services, and networks. On the other hand, even if it is cumbersome to manage this vast amount of data, data also contains valuable information about users, services, and network status, that

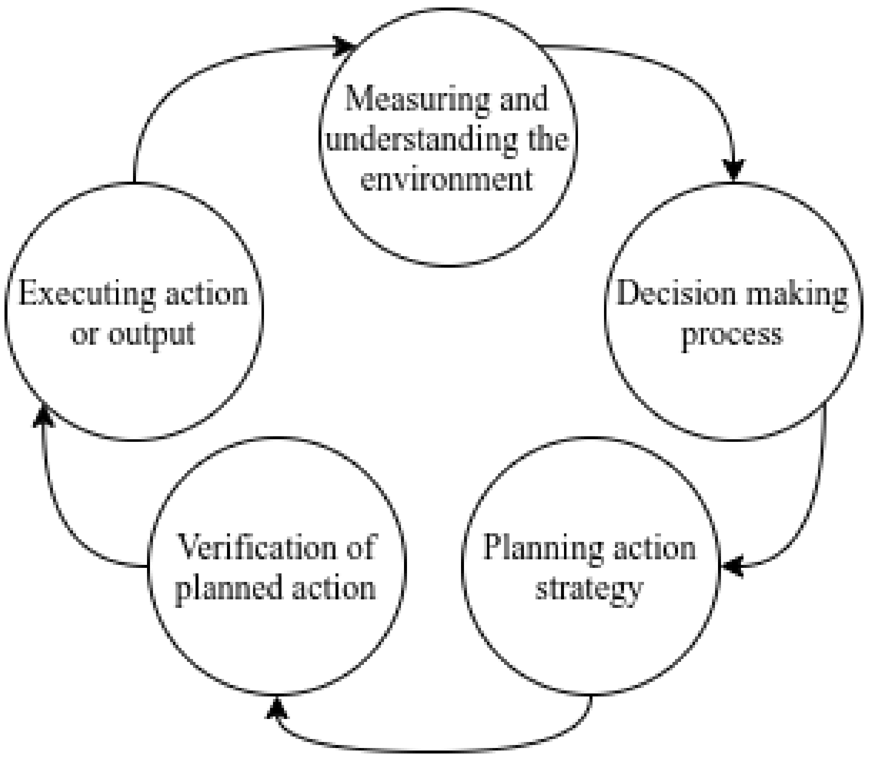

Measuring and understanding the environment
Decision making process
Planning action strategy
Verification of planned action
Executing action or output

FIGURE 1.7: Cyclic process of network management.

could be extracted for effective network management. Information could be used to facilitate the provisioning of services. This could happen through systematic management and analysis of data to extract valuable information to self-manage the network.

Legacy network management systems (NMSs) were mostly implemented in hardware but today, an increasing number of them have been implemented in software. These systems are responsible for the good health of a network [295]. They monitor, maintain, and optimize the network. NMSs provide a number of functionalities such as fault detection, device management, performance analysis. Network management raises new problems, that need to be tackled to ensure a robust and secure networking system is completely achieved. The trend today is that future networks should be more independent, requiring almost no human intervention for healing, protection, optimization, and (re-)configuration. This results in ANM system [278]. It is generally agreed that a greater degree of service knowledge and efficient utilization of network resources would be needed for the next phase of a multi-tenant network.

1.3.1 Network Management and Control as a Cyclic Process

Network management and control is a cyclic process that starts from observing the environment and ends the cycle to acting and changing also influenced by the variations in the environment. Figure 1.7 depicts a general network management process.

- Measuring and understanding the environment is performed through observing and taking some measurements to capture any behavioral change. Various techniques could be employed to observe the environment. These techniques could be independent of the environment to observe and the parameter to be measured. It could be sampling the instance of events or continuous measurement of network events and activities. Observing and measuring the environment provides important information about the status of the network being managed. For instance, measuring the bandwidth across a given link or path would provide valuable information that could be used in the routing decision of new traffic and services. It could also be used to monitor and guarantee the required QoS for a given service.

- Decision-making process is the process of extracting and utilizing valuable information for the final decision. For instance, a decision could be made for routing of new traffic or rerouting of ongoing services by observing the current network state or amount of traffic in a given link or path for optimal or maximum resource utilization depending on the policy stated by the network administrator. The decision-making process is the most critical and complex part of the network management process. Depending on the decision, it may involve performing complex optimization algorithms including ML techniques.

- Planning action strategy is the procedural step needed in order to execute the decisions such as what kind of actions or configurations should be performed on an element to have the required network behavior. For example, once the decision is made to allocate resources, a configuration file needs to be prepared for execution in the network elements or devices. This allocates the required network resources, as agreed upon in the decision-making process.

- Verification of planned action is the process of validating the plan before executing it on the target network elements or devices. These could be configuration files or steps, which have to be verified and checked for accuracy and consistency. It is very important to verify the action as the execution of the action may have undesirable consequences on the target device and/or the network behavior in general. Since the network is a dynamic environment and the process of settling for a given decision may take more time, the decision may be outdated or no longer useful, or sub-optimal depending on the scenarios considered.

- Executing the planned action is done through implementing the verified and accurate final action plan. It could be a configuration path because of a requested service, reservation of end-to-end network resources, service isolation, or a part of QoS provisioning and guaranteeing.

- Finally, monitoring the system behavior is analyzing the effect of the action on the overall system behavior. This could be through loop-back control.

In the decision organization of any network, control loops are a very essential part of a network management to employ automation. They provide information throughout different parts of the network in question thereby facilitating self-adaptation and later, self-management [30]. They facilitate data collection from which decisions are made. However, loop-back control problems arise together with the increase in the cost and complexity of the system. Another challenge exists, in the case of policy-based decisions where conflicts can arise from two policies with similar conditions but different actions.

1.4 Edge Computing, Cloudification, and Containerization

The current technological trends in the research and industry are taking the direction of systems being distributed and loosely coupled. This research journey can be summarized as the move from centralized and hardware-based towards distributed cloud-based and virtualization/softwarized approaches. Before virtualization, applications were installed as a complete system with its OS over some underlying hardware. This, however, had several challenges such as constrained mobility and difficulty running updates. As a result, virtualization came along and offered the possibility of having many applications, each in its virtual environment, but sharing the same hardware infrastructure and resources. This greatly improved configuration and management of applications since VM are easy to clone and install in a different location.

Applications are designed to run in different isolated spaces (e.g containers, dockers, etc) while utilizing the same resources and operating system. The concept of "containerization" [296] comes with a number of benefits such as efficient resource utilization, quick development and debugging of applications, the possibility of fault isolation, and easier management. With the network, automation arises the possibility for any network to self-heal (fault isolation), self-configure in the shortest time possible, self-protect, and self-manage.

Today, implementation is moving from on-premises applications to cloud-based applications [157]. Applications can run at the customer premises i.e. in a server in some server room in a building. This is becoming infeasible in terms of CapEx and OpEx, scalability, and security (against theft and natural disasters like fires). Containerization alone does not imply cloudification. Applications are hosted in the cloud and as long as one has access (and an internet connection), the services are available. This has resulted in inefficient resource utilization, security against theft, and easy disaster recovery due to the availability of an off-site backup. The costs of migrating services and maintenance of equipment are significantly lower than the potential OpEx [229] without virtualization and automation.

1.4.1 Containerization and Cloudificaton

Network automation could employ the concept of distributed computing and centralized control. In a distributed computing the solution to a problem is obtained through the division of tasks amongst a group of network management elements. This results in improved performance, flexibility as tasks can be performed by different machines irrespective of their geographical location and reliability since a glitch in a single element does not mean total denial-of-service in the entire network. Distributed computing is employed in network automation through containerization, where different services of applications are run in different containers while sharing the same OS and other networking resources [296]. This happens in an effort to implement virtualization of services, easy fault isolation, and lower system costs.

On the other hand, centralized network control is built around a single controller that manages all the major network control. Network nodes or workstations connect to the centralized controller and submit their requests to the central controller rather than performing them directly, depending on the required network control functions. The centralized controller manages the flow by programming the devices to perform the required packet routing or management. The controller directs traffic according to forwarding policies that a network operator puts in place. This minimizes manual configurations for individual network devices. The centralized control will have a global information about the network.

While maintaining a global view of the network and centralized control principles, the controller could be designed using a monolithic system or as a decoupled microservices or multi-agent systems. The existing controller systems are designed as monolithic systems, such as Ryu SDN controllers. However, very recently, a microserivce based controller design is proposed by the µONOS projectmicroONOS. It decomposes the controller functions into virtual network functions and deployed them as microservices in containers. The decomposition provides an independent implementation of controller functions. Moreover, this approach provides flexibility in terms of dynamic scaling and backup deployment of functions. Such design provides the option of distributed deployment of the centralized control.

The network functions, that are either be designed as microservice or multi-agents, can be deployed as containers. A container is a lightweight computing package that is abstracted away from the host operating system (OS). OS making it easy to migrate containers from one device to another without the need to adapt them to the new device. The applications deployed in each container, each have their libraries and configuration files. The containerization technology today is divided into:

- Container runtime, a software that executes container images on a given node. These include Docker, Docker Enterprise, CRI-O, rktlet, containerd, Microsoft Containers, etc.
- Cluster management and deployment technology such as Kubernetes, Docker Swarm etc for workload management and resource assignment.
- Storage containers like BlockBridge and EMC / libstorage for storage purposes.
- Container security like Twistlock, Aqua, Aporeto, etc to provide for detection of any anomalies such as intrusion and aid in processes like single sign-on, vulnerability scanning among others.

The distributed nature of computing is employed either on-premises or in the cloud. The paradigm shift today is cloudification due to its advantages such as faster disaster recovery, security, more resources, and easy access at any time regardless of geographical location. Cloud computing can be described as a model enabling the sharing of a variety of computing resources in the form of services through remote access over the internet rather than a physical computer or storage disk. This computer system is split into two parts i.e. front-end (clients or user devices) and back-end (servers).

1.4.2 Edge Computing

Terms such as edge computing and cloud computing have arisen because of the overwhelming need for faster and better technologies. Edge and fog computing systems also bring data processing closer to the source of data generated i.e. the sensors [79]. The key reason for this is to minimize the amount of data sent to the cloud, thus reducing the latency. As a result, the response time of the system particularly improves for applications requiring low latency, such as the 5G URLLC services.

Figure 1.8 illustrates clearly the difference between cloud, fog and edge computing [275]. The cloud layer (cloud computing) can be described as a large and

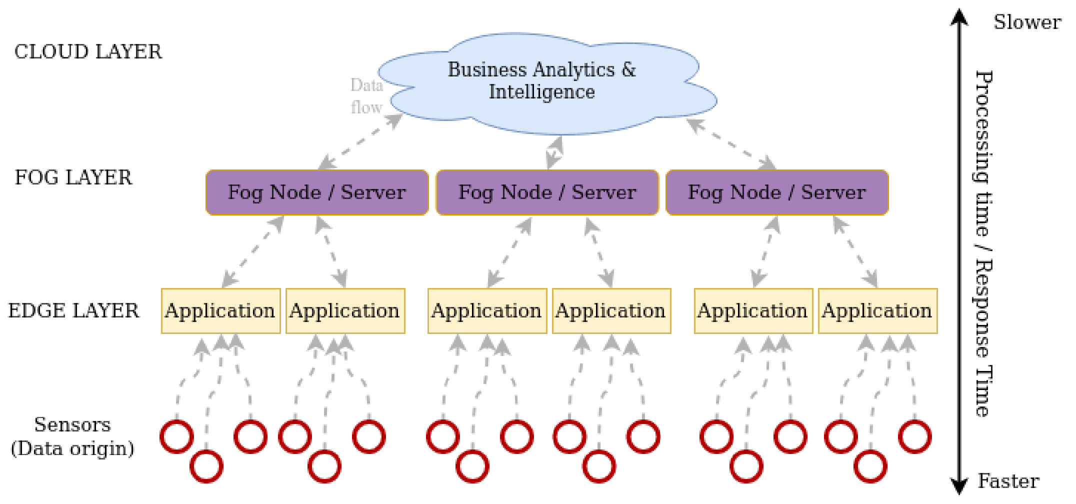


FIGURE 1.8: Cloud computing Vs. Fog computing Vs. Edge computing.

centralized data storage and processing facility, that exists far away from the data source (sensors). Accessing services here means accessing resources on the Internet, so experiencing considerably higher latency. Fog computing, on the other hand, is closer to the information source and is characterized by its distributed nature, low-latency. A number of standards and protocols are required in order to access it. Edge computing (similar to fog computing) happens in the edge and at the devices, where the sensors are placed thereby, resulting in lower latency and faster system response.

1.4.3 C-RAN

Cloud radio access network (C-RAN) is a virtualization paradigm, which aims at moving RAN and baseband functions and procedures to cloud data centers. That would help to reduce power consumption while increasing energy efficiency of heterogeneous RAN management, deployment, and updates.

Figure 1.9 depicts the idea behind C-RAN. Legacy 4G/LTE RAN requires base stations (BSs), which equip a baseband unit (BBU) at each radio site. Nevertheless, this solution is neither scalable nor optimized in large heterogeneous scenarios of future generation networks. On the other hand, by implementing virtual BBUs (v-BBUs), the network achieves higher flexibility in management and configuration of the RAN by detaching baseband processing functionalities from standard BSs; thus, BSs will become pure radio remote heads (RRHs), whereas baseband processing will be moved to dedicated data centers with shared processing facilities. This approach is expected to reduce complexity and power consumption of the RAN. However, the allocation of virtual resources and processing tasks has to be assigned effectively not to increase delays and loads.

In current 4G cellular networks, baseband processing at BBUs3, 4 includes all the processing due to lower layers of 4G protocol stack. The operations of a BBU involve physical layer processing (4G baseband signal processing components include ASICs, DSPs, microcontrollers, and FPGAs), smart antennas, and multiuser detection required to reduce interference, modulation/demodulation, error correction coding (which increases the complexity of the baseband processing at the receiver), radio scheduling, and encryption/decryption of packet data convergence protocol communication (both downlink and uplink). Multicarrier modulation is

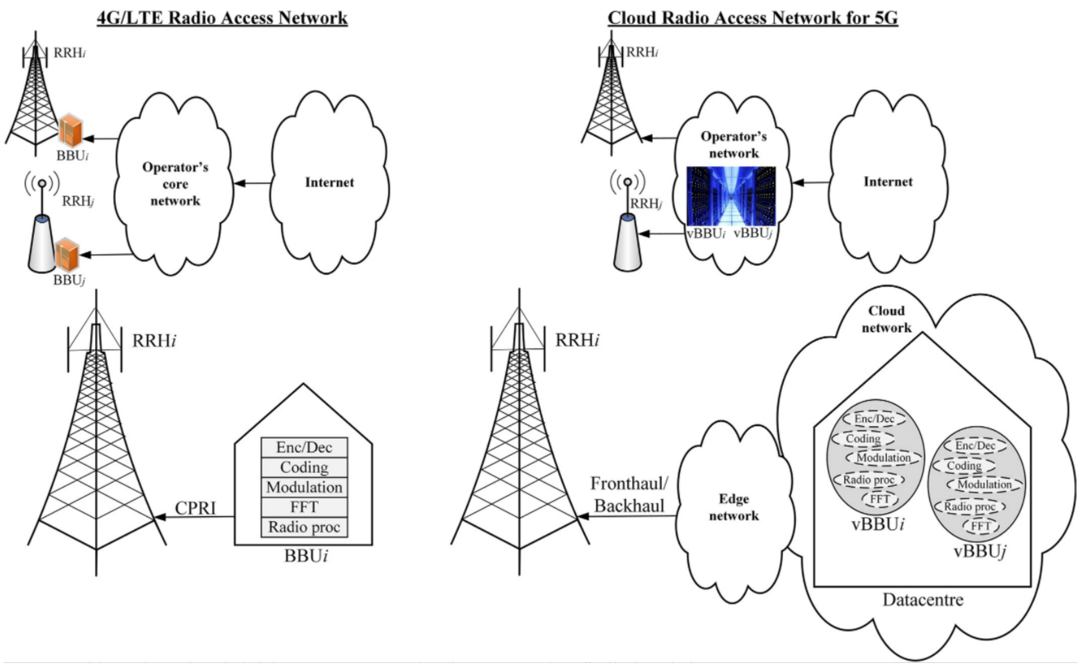


FIGURE 1.9: Downlink communication in heterogeneous 4G or LTE RAN and heterogeneous 5G Cloud RAN. The latter places baseband processing at virtual baseband units (BBUs) in operators' data centers and run them as virtual machines or virtual functions in containers. RRH, radio remote head

also a baseband process. The subcarriers are created using IFFT in the transmitter, and FFT is used in the receiver to recover the data. A fast DSP is needed for parsing and processing the data. Multiuser detection is used to eliminate the multiple access interference present in CDMA systems.

1.5 Microservice and Multi-Agent Systems for Autonomic Networking

Here we will introduce the thee contending technologies for service design. These are monolithic, microservice, and multi-agent.

1.5.1 Microservice Architecture

Microservice is a variant of the service-oriented architecture (SOA) structural style in software development [138]. It arranges an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained decoupled functions. The protocols are interconnecting the decoupled services are lightweight. The architecture describes a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around an organization, business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

As per the above definition, microservices-based systems provide the possibility of building a system from small heterogeneous components. A lot of existing tools could be geared toward microservices such as communication interfaces.

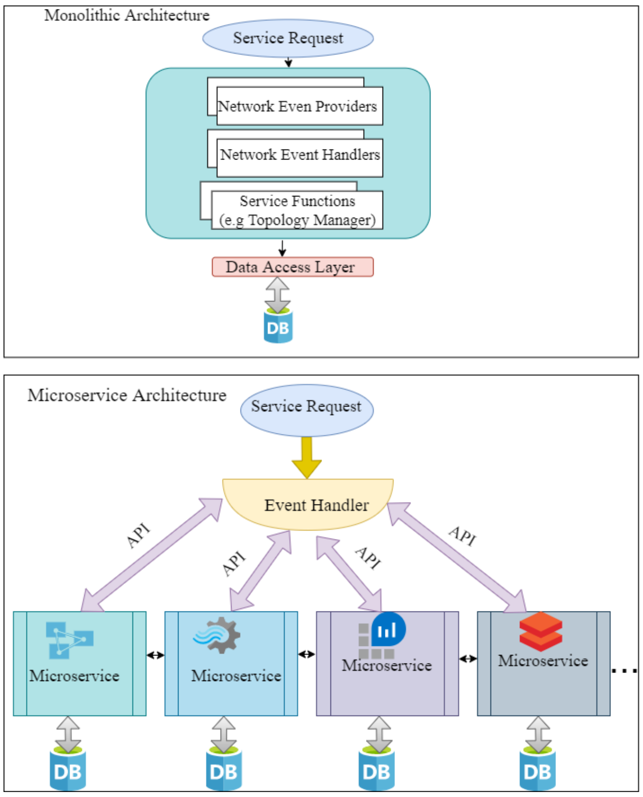


FIGURE 1.10: Microservice Architecture

Microservices-based systems have the advantages of scalability, reusability, flexibility, and agility. First, the functions that we are proposing to be designed as microservice inherit the general microservice properties. Second, the microservice-based function could be considered as separate functions that could be placed in a container and cloud environment. Third, by instantiating and sequencing the necessary functions, we can recreate the decomposed monolithic systems with only the important and required functions hosting them in the nearest edge computing data center. Figure 1.10 depicts the basic microservic architecture.

1.5.2 Application of Multi-Agent for Network Automation

There exist several research works emphasizing the application of the multi-agent-based system in networking. MAS has been used in many areas [232] mainly focusing on works related to networking. We divide them by areas of application e.g. wireless, wireless mesh, security, mobility management, and network management system. A multi-agent-based configuration in a ubiquitous wireless network is presented in [130]. The author showed how agents are used in the RAN where the information is loaded and fed back with the help of communication between various agents. Q–learning technique is used to provide the agent with cognitive abilities to make decisions on when to apply load balancing.

A significant number of works exists utilizing agents for WSN [240, 53, 158, 121, 154, 314, 201, 45]. A multi-agent-based system architecture for WSN is proposed by [121] with four types of agents: interface, regional, cluster, and query agents. In comparison to conventional client/server architecture, mobile agents have many advantages for WSN. In [240], a multi-agent-based hybrid protocol is proposed, utilizing the advantages of combining value and decision fusion. The aggregation is performed to avoid data redundancy before arriving at the sink. Mobile agents perform the task of data aggregation at the nodes rather than at the processing element resulting inefficient bandwidth usage and improved network lifespan. A source grouping along with a tree-based ordering mechanism is proposed in [158]. Recently, the authors in [53] designed a migration route planning algorithm based on mobile agents. Other related works in the area of WSN considered: a multi-agent simulation [154] and performance comparison of multi-agent middle-ware platforms [45]. The mechanism for Cooperation and consensus for multi-agent networked systems is dealt in [201, 314, 55]. Using the tuple space model, in [127], MAS architecture for intelligent home network is presented.

Moreover, some other works focused on more specific applications such as security, service discovery, and service migration. The article in [12] proposed an approach for dynamic service discovery in service-oriented architecture based on a multi-agent system using matchmaking technique. Patri et al. [211] developed a generic migration algorithm derived from a search-based rational agent decision process. Such an algorithm can deal with uncertainties to provide the migration path. The migration is computed by using a maximized utility function. Several works also exist showing the use of intelligent autonomic agents for security applications such as risk assessment [180], and network intrusion detection [208]. The authors in [318] presented mobility management over the satellite networks based on virtual agent clustering. In [120], the authors proposed the use of MAS methodology to facilitate the decentralized reconfiguration of power systems to offer more flexibility and control as well as avoiding the problem of a single point of failure as witnessed in centralized systems. The authors in [166] offer an ASBR model as a solution to DoS attacks in WSN. They describe a model in which a network is able to monitor and identify nodes compromised by DoS attacks and self-recover in time.

None of the works considered an organized management architecture, like in [271, 155, 274], for the overall network system management. Guo-zhu et.al,[271] presented a multi-agent-based management system designed for distributed network. They developed an intelligent management architectural framework. The architecture organizes agents into three types: center layer-agent, region-layer agent, and access-layer agent. The functions of each agent and cooperation mechanisms in the framework are discussed in the article. Some interesting work about control systems is presented in [262] from the control research community. Several works existing related to steady-states of the closed-loop system and study-state analysis of multi-agent-based system [242, 243].

1.6 Artificial Intelligence and Machine Learning for Networking

Currently, there is a shift from manual defined network (MDNs) to intelligent defined networks (IDNs) [43]. Operations, such as optimization and configuration, in MDNs, are completely dependent on human intervention. Whereas the same operations in IDNs are not. The complexity introduced by network automation can be reduced by introducing ML technology into the network [85] thereby making it an intelligent network. Intelligent networks, rather than following strictly defined policies/laws, can learn from past data to guide new predictions or decisions.

On the one hand, through learning from new data, the IDN can dynamically adapt to a changing situation of the network and develop its intelligence with it.

There are various methods to extract information for given data. ML tools such as fuzzy logic, case-based reasoning, evolutionary computing, artificial neural networks, and intelligent agents provide powerful aids in solving difficult applied problems, that are often real-time, involving large amounts of distributed data, and benefiting from complex reasoning. Through sophisticated user interfaces and visualizations, IDN may also strive to help human-based decisions through pre-processing data and providing insights to users and network administrators. Most intelligent networks employ machine learning paradigms to predict/classify network traffic patterns, application patterns and later make decisions based on these predictions and/or classifications. The ML paradigms include: supervised learning, unsupervised learning, and reinforcement learning [43]. These paradigms influence decisions how data is collected, processed, and the inference of decisions from the data.

We continue to highlight some of the networking areas where machine learning paradigms have been applied. An interesting comprehensive survey paper on the application of ML towards network automation is presented in [43]. ML exploits information from the network and users' data to learn and perform the required changes to suit the demands. Recently, using SDN's capability in collecting an enormous amount of packet and traffic data, lots of research has been done to explore the applicability of ML in networking for traffic analysis [223, 181]. In [252], an autonomic VNF placement is discussed considering a three data center hierarchy for workload offloading between the data centers depending on the traffic load. Most of the recent work on ML-based VNF placement has focused on exploring ML for dynamic traffic-aware VNF placements [239, 212]. In [239], the authors presented a method called z-TORCH orchestration mechanism that uses unsupervised learning to monitor VNF key performance indicator (KPI) and reinforcement learning to find a trade-off solution for reliability and complexity of the monitoring system. In [263], the authors discussed a classification approach to dynamic resource scaling and VNF placement. They used a specific version of neural network with multilayer perceptron to predict the required number of VNFs at a given time using a Dataset collected from a commercial mobile network. Time series prediction for VNF placement is presented in [269]. The authors in [269] first analyzed the traffic characteristics of the data center and devised a traffic forecasting technique. Based on traffic forecasting, they develop a deterministic algorithm to determine the VNF resource scaling.

The application of ML in network automation raises a few fundamental questions. To what extent should the network be autonomic or independent of the network administrator? Does it mean zero involvement? If this is the case, then how does the system know what to do at each stage of the network, either it is in transition state or steady-state? These are two extreme scenarios that should be considered. The network elements are assumed to be capable of interacting and learning their working domain to perform a given function and form the overall network. In this sense, they could start with a predetermined set of rules. On the other hand, they could start with no rules at all. In order to realize full network automation, a comprehensive approach for the cumulative decisions considering the overall decision latency, reliability, consistency, and computational efficiency is a necessity.

1.7 LTE, 5G, B5G, and 6G

1.7.1 LTE Technologies

Long-term evolution (LTE) is the fourth generation mobile broadband network that attracted billions of mobile subscribers. There is still a demand that could even increase subscribers. LTE delivered huge improvement over the predecessor with the overall performance. It also paved the way towards more intensive softwarized network functions compared to previous generations. There are different versions of LTE. The most advanced and contemporary versions of LTE are known as LTE-Advanced and LTE-A pro. LTE incorporated inter-cell interference coordination(ICIC), multiple input multiple outputs (MIMO), the separation of control, and data plane known as control user plane separation(CUPS). This helps transition towards next-generation small cell technologies, such as network functional application platform interface(nFAPI). Other fundamental changes are eNB splitting (BaseBand Unit and Remote Radio Head), modification of core network (CN). The last releases of evolved packet core(EPC) have lost circuit-switched mode, allowing all components to communicate only through IP. This enables full packet switching. The main components of 4G EPC are described as follows:

- Mobility management entity(MME): is responsible for management and control of the EPC. It takes care of handover, paging, UE/eNB attachment. It also manages the necessary information to accept or reject access requests to the network.
- Home subscriber server (HSS): this function is responsible for the storage and access of users' information. It allows MME to have persistently archived all identifiers necessary to accomplish identification operations.
- Packet data network (PDN)-gateway (PGW): is fundamental for the correct functioning of EPC. PGW's main job is to allocate and assign IP addresses. It also allows access from EPC to the internet and the external world.
- Serving-gateway (SGW): S-GW is the complementary function of the user plane for P-GW, connecting radio access network (RAN) to EPC and granting access to PDN gateway.
- Policy and charging rules function (PCRF): it mainly manages different policies for user access, allowing differentiation between planes and QoS access.

The above functions follow the legacy deployment of 4G/LTE network, which is mostly hardware-based. This lucks the flexibility that the current network is requiring, especially in the era of network softwarization and cloudification. Moreover, an experimental tesbed is challenging to build since it requires using hardware-based functions.

1.7.2 5G Technologies

The first version of 5G commercialization has started to roll out with softwarization of functions. Its roll-out is facilitated by innovative possibilities derived from the softwarization and reduction of hardware. This enables the innovative solution to be developed to achieve the dynamic and heterogeneous demands of users. Since release 15 of 3GPP, which is the last version of LTE, all the developments are considered as an early version of 5G that links with LTE. Release 15 is more of a transition

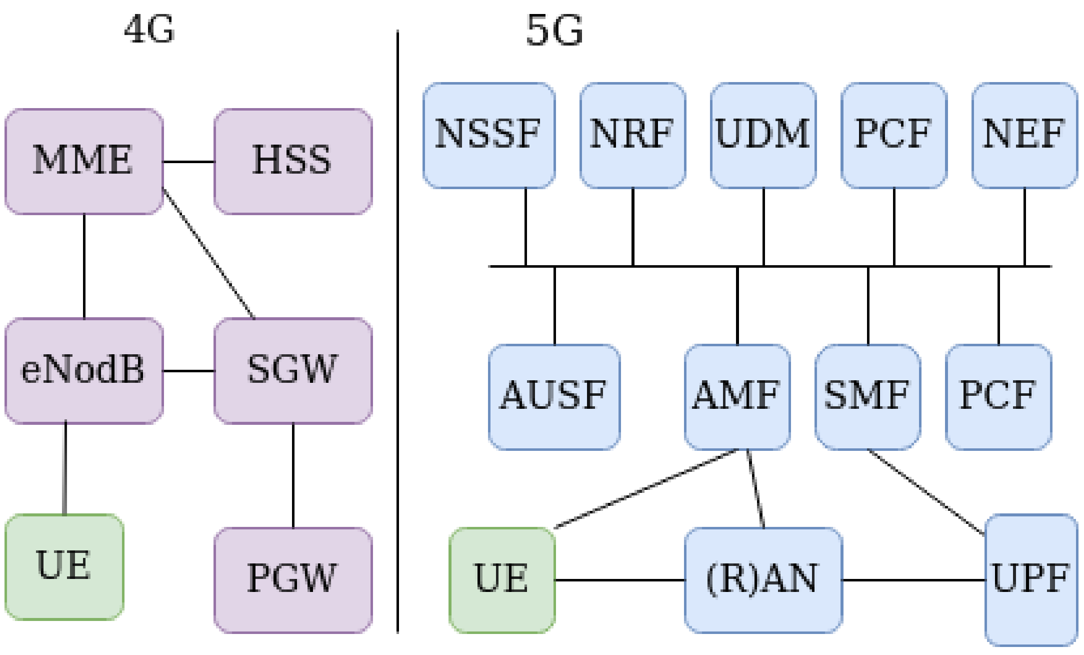


FIGURE 1.11: 4G/5G Core Network

deployment, that allows new features but still being compatible with LTE-A. The major novelties involved in next-generation RAN (gNB) is the fundamental split between central and distributed unit (CU/DU). Moreover, there is also a separation between control and user plane (CUPS) with the dedicated entity for user plane function(UPF). Furthermore, there is an independent design and implementation of services such as RAN and CN. The 5G use service based architecture (SBA), utilizing micro-services for core component design. In general, 5G aimed at introducing the functions based on service-oriented approach, specifically, as microservice and multi-agent approach in service design[14].

Open projects on softwarized emulation of 4G and 5G are an emerging paradigm in the development of 5G. There are several alternatives. For the sake of research, it is preferable to consider open and free projects, which present anyway consistent and advanced opportunities. 5G components could be deployed in a virtual or docker virtualization environment. Examples of such works are OpenAirInterface(OAI) software alliance, Free5gc, Open5gs, SRS-LTE, and Labora projects are among several initiatives aimed at the progress in democratization and open deployment 5G deployment. These are an emulation of 4G/5G components in a containerized environment. The architecture of 4G/LTE and 5G are depicted in 2.27.

5G is expected to address the strange requirements of heterogeneous users. This can be addressed by creating more flexibility in the legacy network, paving the way for innovative solutions such as dynamic service orchestration and network automation[14]. In general, 5G needs to fulfill the requirements of the following categories of services: enhanced Mobile BroadBand (eMBB), consider high speed of end user data and system capacity as crucial, except for a slightly less focus on connection density and latency; ultra-reliable Low latency communications (URLLC), latency along with mobility will be crucial for reliability improvements; massive Machine-Type Communications(mMTC) main concern is the connection density parameter and network energy efficiency.

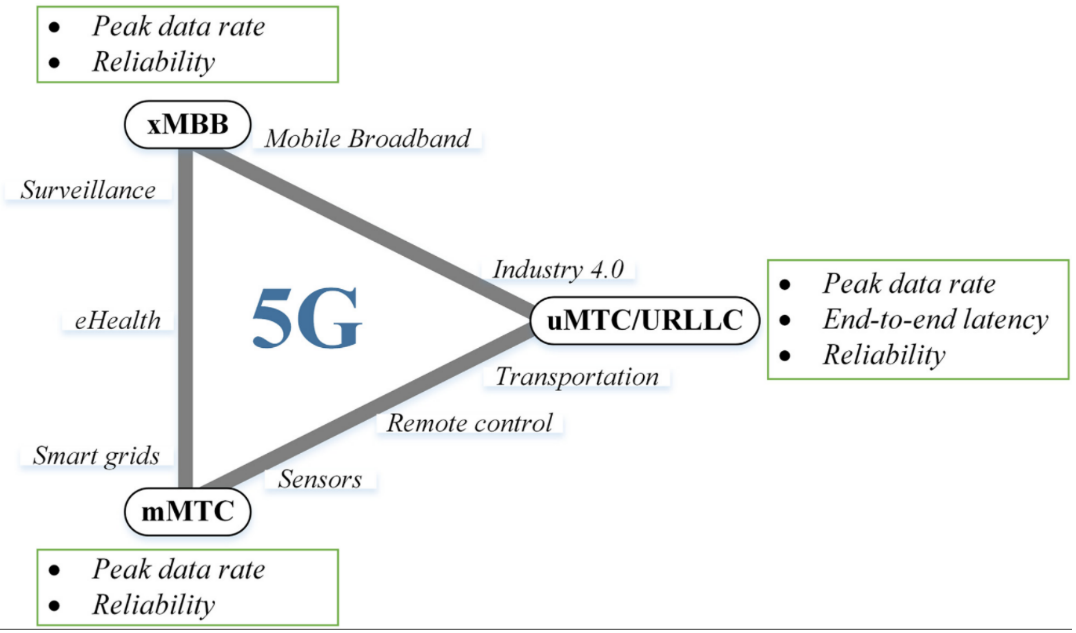


FIGURE 1.12: enhanced mobile broadband (eMBB), ultra-reliable low-latency communications (URLLC), and massive machine-type communications (mMTC)

1.7.3 5G Service Requirements

5G and beyond networks are expected to have a heterogeneous environment made up of not just mobile phones but different devices that support a wide number of applications. Some applications require a large number of devices or simultaneous connections at the same time while others require low latency in order to offer efficiency and reliability. 5G services are divided into three main categories [147] e.g. enhanced mobile broadband (eMBB), ultra-reliable low-latency communications (URLLC), and massive machine-type communications (mMTC). Figure 1.12 shows the three types of 5G services and their requirements.

eMBB services require high data rates. Some of the use-cases include large-scale video streaming and virtual reality. This type of service requires an enhancement of the existing fourth-generation (4G) long-term evolution (LTE) broadband data rate capability. URLLC services include autonomous driving, remote surgery, and industry control with robots. This second category of 5G services targets very low-latency and ultra-reliability since it will support verticals such as remote surgery, automated driving. These services are also referred to as mission-critical services because they will provide an infrastructure for emergency-governmental communications. mMTC services include billions of devices connected at the same time. This third category of 5G services includes devices such as those in the IoT, smart cities, and smart buildings. The main performance requirement for mMTC networks is the need for highly-dense connectivity (about 1 million devices per km^2).

Due to the heterogeneous nature of 5G and beyond networks, stringent service requirements are targeted e.g high data rates, connectivity, and low latency. Thus, management and control of such networks are very difficult, the CapEx increases with the need for connectivity, and more devices or sensors. Moreover, OpEx increases due to the constant need for human intervention to manage the networks. As previously mentioned, this motivates the need for network automation in 5G environments.

1.7.4 B5G, and 6G

Recently, the attention and the effort towards the next generation (6G) has started both in the scientific, public, and private communities. Some consortia have already started working on the definition and characterization of future 6G communication networks, preparing the ground for its standardization by 2030. In Europe, some of the major projects are the EU Flagship Hexa-X [36], the 5GPPP CORDIS RISE-6G [226], and the 5GPPP RIA DEDICAT 6G [72]. In parallel, the US has also started the Next G Alliance [188]. All those results are expected to be contributed towards the ITU Radio communication.

However, various promises made by 5G are still not satisfied and there is the feeling in part of the community that 6G could just be a 5G+. Moreover, network softwarization brought 5G closer to the Internet community, which does not use any 'generation' terminology, but it provides continuous network upgrades and software updates. Next, 5G has already promised support to many verticals, which are still waiting to receive the promised low-latency reliable connectivity and almost continuous service availability. In such a conceptual/technical still 'liquid' and changing situation, this article tries to state some important conceptual, terminological, and technical characteristics and guidelines that will make 6G.

In the last few years, speculations have been proposed to give some shape to the future 6G networks. After the imprint left by 5G, even 6G started from the definition of potential targeted use cases. Among the various proposed verticals, Augmented Reality (AR) and Virtual Reality (VR) have been the main and most popular drivers of the 6G vision. Even if already proposed in a preliminary form during 5G, now AR and VR visions have significantly been extended towards the realization of 3D holographic video representations and interactions. Some initial studies have estimated that a raw hologram, without any compression, with colors, full parallax, and 30 fps, would require $4.32 Tbit/s$. The latency requirement will hit sub-millisecond, and thousands of synchronized view angles will be necessary [165]. Additionally, the idea has also been to make holograms physically perceptible by associating not only audiovisual information but also haptic data as in the Tactile Internet.

Chapter 2

Mathematical Models for Softwarized Networks, VNF Design, and Softwarized LTE/5G Functions' Local Deployment

In this chapter, we focus on the softwarized network analysis through mathematical models, new type of network function design as VNF, and emulation of LTE/LTE-A/5G network functions. The simulation of VNF design and emulation of LTE/5G are performed by MSc students that I supervised.

2.1 Introduction

The current distributed wireless network architecture is unable to support 5G services' high computational demands, stringent requirements and massive number of heterogeneous devices. It requires huge computing power in each cellular sites, which is extremely expensive. Therefore, the proposal is to implement the BBU in software and place it in a data center with large computational power, memory and storage than the existing cell sites (see Figure 2.1). However, by considering only computational demands, a traditional data center would not be the best choice for deploying virtual BBU (vBBU) as it incurs in a significant transmission delay. Therefore, European Telecommunications Standards Institute (ETSI) recommended mobile edge computing (MEC) for vBBU placement to reduce the delay between the user and processing unit. In addition to minimizing the latency, MEC gives some level of distribution of computing power, which provides significant reduction of unnecessary concentration of service traffic to a single data center.

However, moving the vBBU into the edge/MEC, and implementing it in software, requires careful considerations regarding processing, latency, reliability, service admission, network resource utilization, energy consumption, and service throughput. To this end, several works have been published for VNF placement from various perspectives. However, to the best of authors' knowledge, none has considered the trade-off among six parameters simultaneously, while considering service differentiation in VNF placement.

Here we correlates the mathematical modeling in [26] by incorporating a theoretical analysis of VNF placement in an edge data center for C-RAN. Moreover, it enhances the theoretical discussion in [26] not only by studying energy efficiency but also jointly formulating 5G requirements. The results suggest that the proposed modeling, along with the indicated techniques, provides admission of services close to 5G requirements.

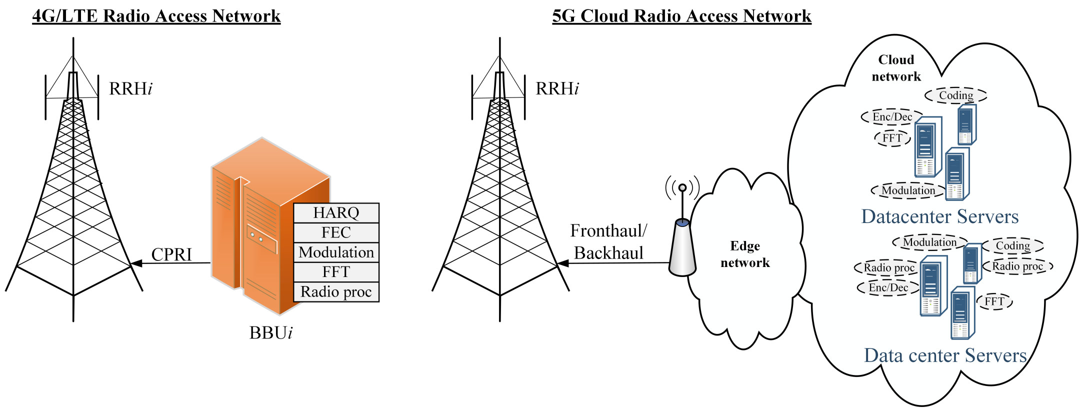


FIGURE 2.1: 4G Architecture, 5G with Cloud Architecture and 5G with Edge Data Center Architecture

2.2 Existing VNF Placement Algorithms

We grouped some VNF placement research works into six categories according to their final objectives.

2.2.1 Resource Optimization

Most related works deal with resource optimization for VNF placement[207, 288, 34]. For example, in [151], the authors considered resource utilization and workload variation in service function chaining (SFC) and VNF placement, with the objective of minimizing the number of physical resources. Similarly, in a recent article [207], the authors presented a spacial resource-demand adaptive algorithm for VNF placement.

2.2.2 Minimizing Service Delay

Another frequent objective in the literature focuses on minimizing end-to-end service latency [169, 305, 58]. However, in most literature, overall quality of service (QoS) constraint is not strictly taken into account, especially in comparison to 5G stringent requirements. More recent works focus on VNF placement, while considering QoS or service level agreement (SLA) [202, 32, 144, 58]. Nevertheless, none of them did account service differentiation for VNF allocation. They only aimed at reducing the delay without considering reliability and energy consumption. In [10], the authors hypothesised that instead of considering how much resource VNFs require to run, it could be possible to look at it as a problem of allocating as much resources as possible for VNFs to meet the service constraints. However, achieving such requirements, using CPU resource over-provisioning techniques, is costly in terms of resources, and energy consumption in edge data centers.

2.2.3 Minimizing Energy Consumption

Few related works focus on energy efficiency in VNF placement [267, 255]. In [131], the authors design a queue-based model for energy cost optimization that minimizes the utilization of physical servers through workload consolidation. In [26], the authors presented a comparative evaluation of energy consumption of BBU in three different scenarios; placing BBU (i) in the cell site as in the existing 4G/LTE architecture, and placing vBBU in (ii) traditional data centers and (iii) in edge data centers.

2.2.4 Maximizing Reliability, Availability and Service Admission

The software implementation of VNFs raises the issue of reliability and fault tolerance [65]. However, reliability and fault tolerance constraint is by far the most overlooked in the literature, in the context of VNF placement in edge data centers. Few works, such as [84, 307, 213, 221, 132], considered reliability and fault tolerant systems, in which they recommend backup VNFs as a solution. In [84], authors suggested that resource over provisioning through parallelization and replication of VNFs, could increase the reliability. Though the solution is pragmatic, it is costly in terms of resources as well as energy consumption in edge data centers. In [150], the authors discussed availability constraint for VNF Placement. They attempted to provide a backup solution with the consideration of resource efficiency in a data center environment.

2.2.5 VNF Placement in Cloud, Fog and Edge Data Center Computing

To this end, several works have been done to model VNF placement in an edge data center, considering its resource constraints [210, 190, 59, 88, 113]. In [294, 175, 319, 114, 212, 269], VNF placement in distributed data centers is considered. The problem of VNF placement in distributed data center is mainly the VNF communication delay between different VNFs, that are placed in geographically distributed data centers. These VNFs are part of a service function chain (SFC) that a given service has to pass through. VNF placement in Fog computing is presented in [57]. It aimed at minimizing the worst completion time of application and applications outage number. In [153], VNF placement in a hybrid data center is discussed. This work focused on modeling hybrid data center as undirected graph where hosts, network and switches are considered. Few works have focused on VNF placement in MEC [142, 67, 185, 305].

2.2.6 Load balancing and Traffic Steering in Service Function Chain

Traffic routing and load balancing aspects are considered in some research [54, 117, 103]. Article [9] focuses on multi-cast service routing with load balancing in VNF placement. They formulate the problem of VNF placement as a cost optimization of VNFs, in terms of resource and cost of inter-VNF links. Some works also consider optimal orchestration of VNFs and steering of traffic through it [313, 270, 5, 122, 11].

In addition to single specific objectives, some other works have focused on formulating a joint objective function to be optimized simultaneously. In [32, 48], the authors presented a heuristic multi-objective optimization taking into account delay and CPU utilization. Whereas, in [134], network link utilization and overall cost minimization is considered. Revenue maximization, while respecting service requirements, is also considered in[174, 77]. In such context, very few research works attempted to combine multiple parameters in their objective function. In [254, 268, 4], the author considered improving service rejection, energy consumption, scalability and operator's revenue. In general, however, there is no work, which jointly considers six main constraints of 5G services.

2.3 C-RAN Modeling as a Multi-layer Loopless-random Hypergraph

2.3.1 System Model

The analysis of techniques to place VNFs in 5G and beyond networks requires a theoretical model, which tries to represent accurately network infrastructure and its characteristics. As previously demonstrated in depth by the authors in [26], existing models, used for network virtualization and specifically for C-RAN, were not able to capture the system-level aspects of the problem. Especially, the evaluation of performance metrics of C-RAN cannot overlook the fact that future generation networks are quite more heterogeneous (consisting of different RANs, interconnected with wired and wireless networks, as depicted in Figure 2.2) and complex than existing and previous wireless cellular networks.

While undirected graphs are an accurate description of existing wired networks, they have limited characteristics to model virtualization of physical network resources at edge and core networks. Moreover, VNFs are hosted at servers in data centers of different sizes and structure. Thus, multilayer graphs are suitable and flexible theoretical models to address this problem [25].

Side by side, in the last decade, stochastic-geometric models were demonstrated to be correct and accurate representations of RAN and wireless backhaul of heterogeneous cellular networks [110, 83]. Even if cellular networks with their base stations (BSs) are already deployed, stochastic geometry allows for a general performance study and evaluation, which is not referred to a specific city or geographical location [83]. In fact, spatial deployment of BSs was demonstrated to follow random point processes (PP). As an example, Poisson point processes (PPP) are an accurate model for BSs in both London and Manchester [161]. Moreover, cellular coverage of BSs is reproduced via Voronoi and multiplicatively weighted (MW) Voronoi tessellations [83].

Given these premises, a random multilayer hypergraph [26] is a mathematical hybrid structure that consists of layers where nodes are placed according to random PPs and deterministic spatial distributions. Moreover, such a mathematical structure can be useful to describe the performances of VNF placement strategies since such evaluations need a system-level end-to-end perspective. The following system model borrows terminology and mathematica description of random multilayer hypergraphs, which was developed in [26] to study latency and energy efficiency of C-RAN. On the other hand, this section also provides and extension of that initial model, in order to make it suitable and accurate for studying techniques to place VNFs dynamically.

Let the future generation network be represented by a multilayer loopless-random hypergraph. Next, let the problem of placing VNFs be formulated by using six main constraints. A multilayer loopless-random hypergraph is a multilayer network, which can describe, via its multilayer structure, end-to-end downlink communication passing through heterogeneous RAN, wireless backhaul, edge and core network and networks of servers (data centers). This scenario is depicted in Figure 2.2.

Let $M = (X_M, E_M, X, L)$ be a multilayer loopless-random hypergraph where

- X is the set of random nodes, which can be placed according to either random point processes (Φ) or deterministic spatial distributions;

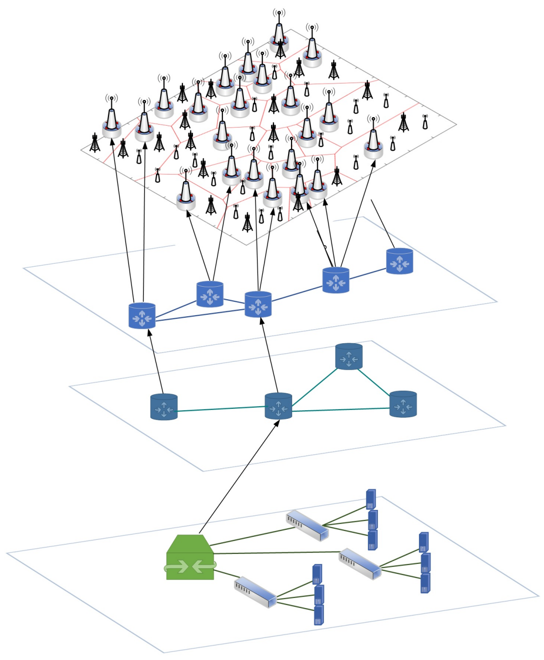


FIGURE 2.2: Physical Infrastructure and Representation of 5G Cellular Network

- X_M is the set of node-layer elements, in which each node in X can differently appear since referred to the respective elementary layer;
- E_M is the set of edge-layer elements, in which each edge can differently appear because it is referred to the respective elementary layer;
- $L = \{L_1, \ldots, L_a\}$ is the set of layers according to the number a of *aspects*; then, subsets $L_i = \{\Lambda_{i1}, \Lambda_{i2}, \ldots, \Lambda_{in_i}\}$ are the sets of elementary layers Λ_{ij}, given the ith aspect; variables $n_1, \ldots, n_i, \ldots, n_a$ represent the number of elementary layers per each aspect.

The total number of layers of M is obtained as $n_{tot} = \sum_{i=1}^{a} n_i$. Next, each elementary layer represents a planar random hypergraph $X = (X_{\Lambda_{ij}}, E_{\Lambda_{ij}})$, where $X_{\Lambda_{ij}} \subseteq X$ and $E_{\Lambda_{ij}} \subseteq E$.

Let $\mathbf{A}_H = (a_{ij})$ be the adjacency matrix of size $|V| \times |V|$ obtained from the ordered vertex set $(<, V)$ of planar loopless hypergraph $H = (V, E)$, where $<$ is a binary relation over V. Next, let $\mathbf{X}_H = (x_{ij})$ be the incidence matrix of size $|V| \times |E|$, referred to the planar loopless hypergraph $H = (V, E)$. The definition of adjacency and incidence matrices of M require the generalisation to tensor theory [143]. Hence, we define the fourth order adjacency tensor and incidence tensor, which can be seen as the four-dimensional arrays $\mathbf{A}_M = (a_{ijk})$ and $\mathbf{X}_M = (x_{ijk})$ (the third dimension is referred to the number of elementary layer). The former (tensor $\mathbf{A}_M$) has size $|V| \times |V| \times |L| \times |L_1| \times \ldots \times |L_a|$, while the latter (tensor $\mathbf{X}_M$) has size $|V| \times |E| \times |L| \times |L_1| \times \ldots \times |L_a|$.

Next, let's define *hyperedge attributes*:

- the function $c \colon E \to \mathbb{R}$, which associates a weight to the edges of M (the capacity of the 'physical' links measured in b/s);

- the function $\tau \colon E \to \mathbb{R}$, which associates a weight to the edges of M (the delay of the edges measured in ms);
- the function $\rho \colon E \to \mathbb{R}$, which associates a weight to the edges of M (the reliability of the edges defined as $\rho = 1 - P_e$, with P_e the failure probability of an edge).

The two functions c and τ allow the definition of two weight matrices referred to M, the *capacity matrix* $\mathbf{C}_G = (c_{ij})$ and the *delay matrix* $\mathbf{T}_M = (\tau_{ij})$. These matrices have the same size of the adjacency matrix $\mathbf{A}_M$. Especially, the delay matrix has members calculated as the sum of two components $\mathbf{T}_M = \mathbf{T}_p + \mathbf{T}^t_{tr}$:

- $\mathbf{T}_p = (\tau_{p\,ij})$ is a matrix of constants, which identifies the propagation delay (dependent on the distance weight between nodes);
- $\mathbf{T}_{tr} = (\tau_{tr\,ij})$ is the matrix of transmission delay, inversely proportional to the available link capacity at time t;
- $\mathbf{T}_q = (\tau_{q\,ij})$ is the matrix of queuing delay.

Next, let's define *nodes attributes*:

- the function $f_{VNF} \colon Q \to X_i$, where Q is the finite set of VNFs; this function assigns each VNF to a specific node in the network.
- the function $f_e \colon X \to \mathbb{R}$, which associates a weight to the vertices (energy consumption of each node in the network).

2.3.2 Service Modeling

First, based on 5G standard, the three main service are characterized to suit our definition of service category. Type 1 services are strictly inelastic so that their requirements must be satisfied to be admitted. They tolerate very low delay in the range of few milliseconds or even less for some services. And that should be maintained during the service life cycle. Moreover, they also require approximately 100% reliability. This type of services is called URLLC: examples are remote surgery and industrial internet of things (IIoT). Type 2 services have softer and relaxed constraints compared to service Type 1. However, if the constraints are not met, there could still be significant loss of users and revenues for violating SLA. These services include online video streaming and virtual gaming. Type 3 services can be defined as elastic, in a sense that their requirements have to be satisfied also allowing for ranges of values instead of strict thresholds. Such type of services includes IoT such as temperature or humidity sensor networks.

A chain of VNFs is assumed to be an ordered subset of queues Q, generated by an ordering function. This ordered chain can be represented by a directed graph G_c. Each node j has associated capacity (CPU, RAM, and Memory).

An *end-to-end service* (Sr_i) is identified by the ith commodity flow with a quadruple parameters $(s_i, \sigma_i, \alpha_i, D_i)$, where $s_i \in S$ is the source (S is the set of sources), $\sigma_i \in \Sigma$ is the sink (Σ is the set of sinks) and α_i is the indicator function $1_{\mathcal{K}} \colon K \to \{0, 1\}$, which assumes value 1 if the commodity k_i belongs to subset $\mathcal{K} \in K$ of commodities with elastic demand set (value 0 means inelastic demand). Then, let D_i be the *demand set*, which defines the requirements in terms of throughput, latency, reliability, processing demand (workload), and service class (priority indicator) for that specific application. If the commodity is elastic $D_i = \{[\tilde{Th}_{min}, \tilde{Th}_{max}], [\tilde{\tau}_{min}, \tilde{\tau}_{max}], [\tilde{\rho}_{min}, \tilde{\rho}_{max}], L_{ij}, \beta\}$.

The first three requirements are ranges that represent services' Type 2 and Type 3, otherwise inelastic cases which represents service Type 1. The set becomes $D_i = \{\tilde{c}, \tilde{\tau}, \tilde{\rho}, \beta\}$, with constant first three members. The priority β is a value in the range $[0, 1]$, which is used to classify the serving priority of the commodity; the sum of all the priorities is 1. The coexistence of multiple services is called a multi-commodity flow and $K = \{k_i\}$ (with $i = 1, \ldots, m$) is the set of m commodities.

2.3.3 System Model Formulation

Using the above system model, we first formulate the objective functions using the constraint parameters while considering flat service scheduling (without service differentiation). In the objective function, we balance between complexity and comprehensiveness. For practical applicability, simplistic formulation would not be accurate to approximate system's behavior. On the other hand, providing an exact formulation of such complex system would be unbearably intricate with a diminished pragmatic applicability. This is because exact formulation must consider all parameters with probabilistic modeling, such queuing dependence and topological variation per operator, for each node and in each layer of the hypergraph. Therefore, in the following formulation, expectation value of the parameters is considered with a typical topological scenario.

We assign delay, reliability, throughput, service workload, energy consumption and service admission constraint to each node and link. In that sense, the average and expectation measurement metrics are used in the formulation of RAN, microwave (MW) backhaul, wired (fiber) backhaul and edge data center network. However, the expectation value of the parameters at the servers of the edge data center are calculated considering multiple queues for VNF placement and scheduling. Note that the arrival process in a cascaded queue and a sequence of VNF nodes is assumed to be the same. Therefore, in the model, the dependence of service processing and service transmission between cascade VNFs is eliminated by approximating the arrival process of each flow at the subsequent VNF (queues) as a Poisson process. This enable us to employ an M/M/1 queuing model to calculate the average measurement metrics. This approach was first proposed as the assumption of independence in Kleinrock's seminal works on packet switching network modeling using queuing theory. As per the Kleinrock's independence assumption every time a packet is received at a given node in a network, it can be assumed as an exponential distributed that, for example, can be used to generate a new length for the specific packet. Nevertheless, this is obviously inaccurate because packets maintain their lengths as they progress through the cascaded network of queues, but Kleinrock proved that the effect is negligible. Therefore, it should be noted that the assumption of independence between subsequent VNF's may not always be accurate, specifically in the case of traffic mixing, strong positive correlation between service and interarrival time[214]. The exact character and formulation of cascaded VNF's is highly affected by different aspects such as the presence of different traffic classes with specific traffic characteristics which share the same queue, the presence of long range dependence in traffic, the presence of links with different link utilization and the presence of a large number of traffic sources sharing the network. Moreover, our initial assumption of cascaded VNF as cascaded queue may need to be numerically verified. That means the use of more accurate models to approximate cascaded independent queues for VNF sequence could be the scope of future work.

In [316], the authors model C-RAN in a single tier cellular network considering mobile users and BS's as randomly distributed according to two Poisson point processes. We assume each tire as an independently distributed cellular network. Since the sums of two independent Poisson random variables are also Poisson random variables. In our formulation, a three tier access networks with users and base stations distributed according to PPP distribution is considered [316]. As previously described in Section 2.3.1, PPP represents an accurate and reasonable description of users' and BSs' spatial distributions, capable to model various real geographic scenarios. The distribution of overall arriving traffic workload to the edge data center in 24 hours is shown in Figure 2.10.

A given user u_{ij}, from the j^{th} tier, generates a computational workload demand in Giga Operations Per Second (GOPS), which is given by[298]:

$$L_{ij} = \left[30A + 10A^2 + 20\frac{MCL}{6}\right]\frac{R}{50} \tag{2.1}$$

where A is the number of antennas, M is the modulation scheme, C is the code rate, L the number of spatial MIMO-layer, and R is the number of resource blocks.

Let M_f, N_p, and O_m be the number of users, which requested service workload L_{ij}, at a given time, respectively from femto, pico and micro cells. The service workload is assumed to be Poisson distributed. Therefore, the total service workload arrival rate (λ_{tot}), from all the three tiers is given by:

$$\lambda_{tot} = \sum_{i=1}^{M_f}\lambda_i + \sum_{j=1}^{N_p}\lambda_j + \sum_{k=1}^{O_m}\lambda_K \tag{2.2}$$

Similarly, the total aggregated throughput Th_{tot} (in Mbps) from all the three tiers, which is also assumed to be Poisson distributed, is given by

$$Th_{tot} = \sum_{i=1}^{M_f} Th_i + \sum_{j=1}^{N_p} Th_j + \sum_{k=1}^{O_m} Th_k \tag{2.3}$$

End-to-end Service Latency

We first formulate the end-to-end latency of the system, which is the overall latency experienced by a given service. For C-RAN, the end-to-end latency experienced by the service is the sum of the delays form source (user) to sink (edge data center) which is given by:-

$$\tilde{\tau}_{tot} = \tilde{\tau}_{RAN} + \tilde{\tau}_{bh} + \tilde{\tau}_{Edc} \tag{2.4}$$

Where $\tilde{\tau}_{RAN}$ is the latency on the link between RRH and UE, $\tilde{\tau}_{bh}$ is the propagation delay on the backhaul network links and $\tilde{\tau}_{Edc}$ is the delay experienced in the edge data center. The first part of the equation consisting of RAN and backhaul delays are discussed in our previous article [26]. We now break up and elaborate the delay due to edge data center. It consists of data center network delay, processing delay, process queuing delay at each VNF and virtualization overhead delay. In the analysis, virtualization overhead delay is neglected for the sack of simplicity. Therefore, the total delay in a generic edge data center is given by

$$\tilde{\tau}_{Edc} = \tilde{\tau}_{dc-ntk} + \tilde{\tau}_{proc} + \tilde{\tau}_{queue} \tag{2.5}$$

The data center transmission delay is due to the physical link and virtual link delay. This links interconnect the hosts containing VNF sequences required for a given service to pass through. Since propagation delay is not dependent on the incoming service workload, it is reasonable to assume $\tilde{\tau}_{ntk}$ as constant given both equal distance between hosts and equal link capacity c_i, thus imposing a link delay δ_i. The total delay experienced by a given user's service, traversing the sequence of VNFs' with K interconnecting links, is given by

$$\tilde{\tau}_{dc-ntk} = \sum_{i=1}^{K} \delta_i \tag{2.6}$$

The processing delay ($\tilde{\tau}_{proc}$) is variable depending on the given service workload (L_{ij}), serving rate of each VNFs (U_{vnf}) from a group of VNFs (i.e. a so called service function chain) and number of VNFs N_{vnf} of a given type. The dependencies are given by

$$\tilde{\tau}_{proc} = \sum_{i=1}^{N_{vnf}} \frac{L_{ij}}{U_{vnf}} \tag{2.7}$$

The queuing delay ($\tilde{\tau}_{queue}$) is calculated assuming M/M/1 queuing discipline. The total process queuing delay is dependent on the arrival rate, serving rate of each VNFs and the number of VNFs of a given type. Moreover, to calculate the total delay in passing through the required sequence of VNFs, we sum all the delays incurred by the given service, in the service function chain (SFC), assuming a queue at each stage of the VNF sequence. This is given by

$$\tilde{\tau}_{queue} = \sum_{i=1}^{N} \frac{1}{M_{sfc} U_{vnf} - \lambda} \tag{2.8}$$

where M_{sfc} is the processing capacity to host a VNFs sequence (the number of available VNF types that a service require (N) multiplied by each VNFs capacity U_{vnf}), ignoring VNFs' overhead CPU requirements. That means all the available capacity of the data center would be used for service processing (without any other processing overhead such as VNF instantiating), given by

$$M_{sfc} = \frac{CPU_{tot}}{N_{vnf}} = U_{vnf} \times N_{vnf} \tag{2.9}$$

Therefore, to meet the end-to-end delay for a service that is constrained to pass through a given sequence of VNFs, the data center delay is the sum of SFC decision delay (overhead), queuing delay, processing delay at each VNF, and interconnecting links delay (virtual links) between hosts, so that

$$\tilde{\tau}_{Edc} = \sum_{i=1}^{K_{link}} \delta_i + \sum_{i=1}^{N_{vnf}} \frac{L_{ij}}{U_{vnf}} + \sum_{i=1}^{N_{vnf}} \frac{1}{M_{sfc} U_{vnf} - \lambda} \tag{2.10}$$

where K_{link} is the number of links interconnecting different sequence of VNFs in the SFC. The above equation shows the total delay at edge data center. Therefore, the overall end-to-end delay is given by

$$\tilde{\tau}_{tot} = \tilde{\tau}_{RAN} + \tilde{\tau}_{bh} + \sum_{i=1}^{K_{link}} \delta_i + \sum_{i=1}^{N_{vnf}} \frac{L_{ij}}{U_{vnf}} + \sum_{i=1}^{N_{vnf}} \frac{1}{M_{sfc} U_{vnf} - \lambda} \tag{2.11}$$

Finally, the end-to-end delay constraint for a given service is given by

$$\tilde{\tau}_{max} > \tilde{\tau}_{tot} \tag{2.12}$$

End-to-end Service Reliability

Considering our multilayer random hypergraph modeling, end-to-end reliability is the reliability of interconnecting nodes in each sub-layer. This involves nodes from source of an incoming service to the sink. The reliability of nodes up to the data center gateway, which are the network nodes including the gateway router has been discussed in most literature [126]. Therefore, even if all parts of the network are considered in our analysis, here, we only elaborate the reliability of the servers hosting the VNF chain. The considered VNF chain is for a given service constrained to traverse through, including the interconnecting links. The overall reliability is calculated by considering the reliability of each sublayers to be mutually independent events[109]. Therefore, the end-to-end reliability constraints becomes

$$\tilde{\rho}_{tot} = [(\tilde{\rho}_{RAN})(\tilde{\rho}_{bh})(\tilde{\rho}_{Edc})] \tag{2.13}$$

A given physical server's reliability is given by [18, 39]

$$\tilde{\rho}_{phs} = \frac{MTBF_{phs}}{MTBF_{phs} + MTTR_{phs}} \tag{2.14}$$

Similarly, a given VNFs reliability is given by [18, 39]

$$\tilde{\rho}_{vnf} = \frac{MTBF_{vnf}}{MTBF_{vnf} + MTTR_{vnf}} \tag{2.15}$$

Therefore, the overall reliability of edge data center (EDC) is given by

$$\tilde{\rho}_{Edc} = [(\Pi_{i=1}^{N}\tilde{\rho}_{vlink})(\Pi_{i=1}^{K}\tilde{\rho}_{phs})(\Pi_{i=1}^{N}\tilde{\rho}_{vnf})] \tag{2.16}$$

End-to-end reliability constraint is given by

$$\tilde{\rho}_{tot} > \tilde{\rho}_{min} \tag{2.17}$$

Edge Data Center's Computing Capacity Constraint

For simplicity, we only consider the CPU representing the computational constraint in the data center. We assume the equivalent memory and storage is allocated for the required number of VNFs' to function properly. The total capacity of a given edge data center is the sum of all the available (N_s) server's CPU capacity denoted as CPU_{tot}, which is given by

$$CPU_{tot} = \sum_{i=1}^{N_s} CPU_i \tag{2.18}$$

Alternatively, if we assume all servers having equal amount of processing capacity

$$CPU_{tot} = CPU_i N_s \tag{2.19}$$

The total incoming service workload, needed by all the three tiers at a give time, is calculated as

$$\lambda_{tot} = \sum_{i=1}^{M_f} L_i + \sum_{j=1}^{N_p} L_j + \sum_{k=1}^{O_m} L_k \tag{2.20}$$

Therefore, the capacity constraint for an edge data center is given by

$$\lambda_{tot} < CPU_{tot} \tag{2.21}$$

End-to-End Power Consumption

As presented in [26], the total power consumption of cloud RAN with an edge data center (e.g hosting vBBU) in 5G networks will be the sum of the power consumption of all the three main components including: RAN ($\tilde{P}_{RAN}$), backhaul(or fronthaul) ($\tilde{P}_{bh}$), and $\tilde{P}_{Edc}$ edge data center. It is given by

$$\tilde{P}_{tot} = \tilde{P}_{RAN} + \tilde{P}_{bh} + \tilde{P}_{Edc} \tag{2.22}$$

The RAN and backhaul power consumption are formulated and discussed in our previous work [26]. Here, we recall those formulations. RAN power consumption is the sum of all base stations' power consumption, which is given by

$$\tilde{P}_{RAN} = \sum_{n=1}^{N_{cell}} P_{bs} \tag{2.23}$$

where P_{bs} is formulated as [17]

$$P_{BS} = N_{trx}((1-\eta)P_{bs-idle} + \eta\Delta_p P_{bs-max}) \tag{2.24}$$

where N_{trx} is the number of transmission chains. In other words, it is the ratio of transmitting and receiving antennas per site, $P_{bs-idle}$ is the idle power consumption of a base station, Δ_p is workload's dependent power consumption's slope, P_{BS-max} is the maximum RF output power at maximum workload, and η is the fraction of workload variation.

Backhaul power consumption is given by

$$\tilde{P}_{bh} = \sum_{n=1}^{N_{cell}} P_{sw}^{n} + N_{Ante}^{n} P_{link}^{n} \tag{2.25}$$

The data center's power consumption is composed of server's power consumption and switches' power consumption and it is given by

$$\tilde{P}_{Edc} = \tilde{P}_{Edcsw} + \tilde{P}_{Edcsr} \tag{2.26}$$

The server's power consumption is dependent on the incoming total workload λ_{tot} (in GOPS) for i number of users or services belonging to all the three tiers. The power consumption is then given by

$$\tilde{P}_{Edcsr} = \tilde{P}_{idl} + (\tilde{P}_{max} - \tilde{P}_{idl})\lambda_{tot} \tag{2.27}$$

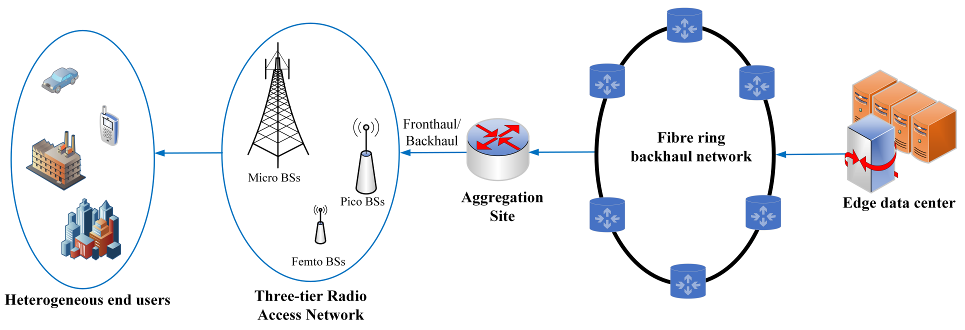


FIGURE 2.3: RAN, Backhaul and Edge Data Center

Similarly, the switches' power consumption is dependent on the incoming load as given below

$$\tilde{P}_{Edcsw} = \tilde{P}_{idl-sw} + (\tilde{P}_{max-sw} - \tilde{P}_{idl-sw})\lambda_{tot} \tag{2.28}$$

Therefore, the total power consumption of C-RAN is given by

$$\begin{aligned}\tilde{P}_{tot} = \sum_{n=1}^{N_{cell}} [N_{trx}((1-\eta)P_{bs-idle} + \eta\Delta_p P_{bs-max})] \\ + \sum_{n=1}^{N_{cell}} P_{sw}^n + N_{Ante}^n P_{link}^n \\ + \tilde{P}_{idl-sr} + (\tilde{P}_{max-sr} - \tilde{P}_{idl-sr})\lambda_{tot} + \tilde{P}_{idl-sw} \\ +(\tilde{P}_{max-sw} - \tilde{P}_{idl-sw})\lambda_{tot}\end{aligned} \tag{2.29}$$

Throughput Constraint

A given user from one of the three tiers generates service request imposing a given amount of throughput demand, which is denoted by $\tilde{Th}_{i-min}$ and $\tilde{Th}_{i-max}$. As per the International Telecommunication Union's (ITU) standard (IMT-2020 5G requirement objectives), the expected theoretical maximum downlink peak data rate is 20 Gbps and the user's experienced data rate is $Th_{i-max}^{\tilde{}} = 100$ Mbps ($100 - 200$ Mbps to support applications such as augmented reality) [69, 28]. Moreover, since the user/service arrival rate is considered as Markovian Poisson process, it is consistent to assume the overall throughput as random Poisson distribution [316]. For a service to be admitted, its end-to-end average throughput demand has to be satisfied in all the subsequent layers of the hypergraph (RAN, backhaul, and edge data center) of the C-RAN. We formulate the service throughput constraint based on the total average throughput in each layer.

In a typical implementation scenarios, backhaul network varies depending on multiple factors such as the network areas that the operator would like to cover, and operators' preference from business point of view. Some operators, specially in the urban areas, may have multiple stage backhaul aggregation links, while others may have single link, to reach an aggregation point. Therefore, we consider a simplified case where the RAN is taken as the aggregation point for active users. Whereas, for backhaul network, we consider two layers assuming maximum of one microwave backhaul layer and one fiber backhaul aggregation layer. Figure 2.3 shows the stages of the network connecting RAN, MW and fiber backhaul and an edge data center.

In this scenario, the fiber network interconnects various aggregation points to the edge data center through a ring fiber links. Each layer of the hypergraph have a hierarchical network where links have different capacity to accommodate various service traffic aggregations levels. Depending on the architectural layers in the hypergraph, the network capacity may vary. The total average throughput is the superposition of all active users' throughput in a given link or aggregation port [162]. For example, if we consider the three tier base stations, the total throughput is the sum of all active users' throughput from all the three tier base stations. This is given below for M_u active users:

$$Th_{bs} = \sum_{i=1}^{M_u} Th_i \tag{2.30}$$

Similarly, the overall service aggregation at a traffic aggregation link from the three tiers is formulated as the sum of M_f, N_p, and O_m active users, from all the three tiers, and N_{cells} number of active base stations. Moreover, we are considering two stage backhaul aggregation layer to approximate typical cellular network. Microwave backhaul are assumed to aggregate base station traffic that are further away from the fiber links (metro or ring network). Moreover, fiber backhaul is assumed to aggregate all user traffic from a given area. The fiber backhaul includes a metro in an urban area that interconnects the aggregation points with the edge data center. The throughput at a microwave backhaul network aggregation point is given by:

$$Th_{bh-mw} = \sum_{n=1}^{N_{cell}} (\sum_{i=1}^{M} Th_i + \sum_{j=1}^{N} Th_j + \sum_{k=1}^{O} Th_k) \tag{2.31}$$

Moreover, the throughput at a fiber backhaul network aggregation point, assuming a single path to the edge data center, is given by

$$Th_{tot} = \sum_{n=1}^{N_{agg}} Th_{bh-mw} \tag{2.32}$$

where N_{agg} is the number of aggregation switches (sites). A service could not be admitted if the minimum throughput demand of the service is not satisfied by the available throughput at each point of the C-RAN. In other words, for system's stability, we have to satisfy multiple constraints, which are RAN capacity, backhaul capacity and edge data center network capacity constraints. The RAN network capacity constraint is then given by

$$Th_{bs} < Lc_{RAN-max} \tag{2.33}$$

The throughput constraint at a microwave backhaul network aggregation point is given by

$$Lc_{bh-mw} > Th_{bh-mw} \tag{2.34}$$

Similarly, the throughput constraint at a fiber backhaul network aggregation point

$$Lc_{bh-fiber} > Th_{tot} \tag{2.35}$$

Edge data center aggregate throughput constraint is similar to the fiber backhaul. However, we presented it for the sack of convenience:

$$Lc_{tot-edc} < Th_{tot} \tag{2.36}$$

For the edge data center network, we consider a three tier architecture, which has both physical and virtual network. Thus, for each path a service is assigned to (from one VNF M_v in a given physical node N_s to another VNF in another physical node), the admitted service throughput demand must be satisfied at the edge data center network links. This is given by

$$Lc_{edc} > \sum_{i=1}^{N_s} \sum_{j=1}^{M_v} a_{ij} Th_{ij} \tag{2.37}$$

a_{ij} is a binary variable indicating if the service is assigned that particular link.

Overall Service Admission Probability

The overall service admission probability of C-RAN system is defined as the ratio between total admitted services and total requested services

$$Ap_{C-RAN} = \frac{\sum_{i=1}^{N_{sr-Adm}} Sr_i}{\sum_{i=1}^{N_{sr-adm}} Sr_i + \sum_{i=1}^{N_{sr-rej}} Sr_i} \tag{2.38}$$

The value of Ap_{C-RAN} is dependent on the composition of the constraints, imposed by all the constituent nodes and edges of the hypergraph, that the service is required to traverses through. In other words, for a given service to be admitted to an edge data center, all services' demands have to be satisfied. That is the admission of a given service is constrained by the possibility of meeting the demand set ($D_i = \{[\tilde{Th}_{min}, \tilde{Th}_{max}], [\tilde{\tau}_{min}, \tilde{\tau}_{max}], [\tilde{\rho}_{min}, \tilde{\rho}_{max}], L_{ij})$ of a given service at the time and point of service arrival. Here, we need to have a measure of service admission for C-RAN system to allow for a comparison among different techniques. Therefore, we can reasonably assume that the conditions of satisfying latency constraint, reliability constraint, computing constraint, and throughput constraint for a given service as independent events, even if the events are not completely mutually independent. With this assumption, the overall service admission probability of the edge data center could be formulated as the product of the probability of fulfilling end-to-end services reliability constraint, the probability of fulfilling end-to-end services latency constraint, the probability of fulfilling services throughput constraint and the probability of fulfilling total computing capacity, as formulated by

$$\begin{aligned} Ap_{C-RAN} = (Pr_{end-to-end-reliability}) \\ (Pr_{end-to-end-latency<\tilde{\tau}_{max}}) \\ (Pr_{total-capacity})(Pr_{end-to-end-throughput}) \end{aligned} \tag{2.39}$$

where admission probability due to latency constraints is given by

$$Pr_{end-to-end-latency} = Pr(\tilde{\tau}_{max} > \tilde{\tau}_{tot}) \tag{2.40}$$

Similarly, admission probability due to reliability constraint is also given by

$$Pr_{end-to-end-reliability} = Pr(\tilde{\rho}_{tot} > \tilde{\rho}_{min}) \tag{2.41}$$

And admission probability due to overall edge data center processing capacity constraint is written as

$$Pr_{Capacity} = Pr(\lambda_{tot} < CPU_{tot}) \tag{2.42}$$

We include the RAN, the two stage backhaul network, and the data center network as sequence of queues in calculating service admission. The service admission formulation considers the service throughput constraints. The queuing sequence assumption enables us to apply Burke's queuing theorem. This gives us the possibility of treating each event as independent events. Therefore, admission probability due to throughput constraint becomes the product of the RAN throughput constraint, the backhaul throughput constraint and the edge data center throughput constraint.

$$Pr_{RAN-th} = Pr(Th_{bs} < Lc_{RAN-max}) \tag{2.43}$$

The backhaul admission probability due to throughput constraint is the capacity constraint in both level of the backhaul networks. At the microwave backhaul we have:

$$Pr_{bh-mw} = Pr(Lc_{bh-mw} > Th_{bh-mw}) \tag{2.44}$$

Similarly, at the fiber backhaul network aggregation point

$$Pr_{bh-fiber} = Pr(Lc_{bh-fiber} > Th_{tot}) \tag{2.45}$$

The admission probability of the edge data center is dependent on the data centers' network aggregation links' capacity, interconnecting link capacity (virtual and physical), overall processing capacity, overall data center processing latency, and overall data center reliability. This is calculated by considering the minimum set of nodes, physical (physical servers) and virtual nodes (sequence of VNFs) and links (interconnecting the VNFs).

$$Pr_{dc-th} = Pr(Lc_{dc} > \sum_{i=1}^{M}\sum_{j=1}^{N} aTh_{ij}) \tag{2.46}$$

The overall admission probability, due to throughput constraint, is combined by considering mutually independent events for the throughput constraints of all part of the network[109] as

$$\begin{aligned} Pr_{end-to-end-throughput} = (Pr_{RAN-th})(Pr_{bh-mw}) \\ (Pr_{bh-fiber})(Pr_{dc-th}) \end{aligned} \tag{2.47}$$

Substituting the above constraint equations for latency (Equation (40)), reliability (Equation (41)), CPU capacity (Equation (42)), and network capacity at each stage of C-RAN (RAN (Equation (43)), microwave backhaul (Equation (44)), fiber backhaul (Equation (45)), and edge data center network (Equation (46))) in a single equation gives us

$$\begin{aligned} Ap_{C-RAN} = Pr(\tilde{\tau}_{max} > \tilde{\tau}_{tot})Pr(\tilde{\rho}_{tot} > \tilde{\rho}_{min}) \\ Pr(\lambda_{tot} < CPU_{tot})Pr(Th_{bs} < Lc_{RAN-max}) \\ Pr(Lc_{bh-mw} > Th_{bh-mw})Pr(Lc_{bh-fiber} > Th_{tot}) \\ Pr(Lc_{dc} > \sum_{i=1}^{M}\sum_{j=1}^{N} aTh_{ij}) \end{aligned} \tag{2.48}$$

Overall Multi-Objective Formulation

In this subsection, we combine the above equations into a single multi-objective formulation that represents the constraint at each stage of the network. Therefore, the overall formulation, that contains the 5G six constraints, is given by

$$Max(Ap_{C-RAN}), \tag{2.49a}$$
$$Min(\tilde{P}_{tot}), \tag{2.49b}$$

while maintaining total CPU constraint:

$$CPU_{tot} > \lambda_{tot}, \tag{2.49c}$$
$$\tilde{\rho}_{tot} \geq \tilde{\rho}_{min}, \tag{2.49d}$$
$$\tilde{\tau}_{tot} > \tilde{\tau}_{max}, \tag{2.49e}$$
$$Th_{RAN-max} < Lc_{RAN}, \tag{2.49f}$$
$$Th_{bh-mw} < Lc_{bh-mw}, \tag{2.49g}$$
$$Th_{bh-fiber} < Lc_{bh-fiber}, \tag{2.49h}$$
$$Lc_{dc} > \sum_{i=1}^{M} \sum_{j=1}^{N} aTh_{ij}, \tag{2.49i}$$

The above formulation is an NP hard problem. Therefore, we try to simplify the method to reach good results by considering the constraint equations in a separate manner and not applying them at the same time. Instead of trying to identify the exact solution for the NP hard problem, we used a simpler approach to maximize the admission probability, while at the same time aiming at satisfying the reliability and latency constraints for ultra-reliable low-latency services. Let us first focus on latency and reliability, which are related to CPU utilization in our next formulation. Since 5G needs to support services with high reliability and low latency, we proposed to improve these two parameters by using backup VNFs to increase reliability of SFC and CPU over-provisioning to reduce latency. We then reformulate the problem incorporating the proposed improvement methods. This is discussed in the next subsection. Using reliability and latency improved methods, it increases the admission of services, which could have been rejected due to these two constraints. However, the methods increase CPU resource usage and energy consumption. The increase in CPU resource usage means that more services could be rejected because of CPU constraints. Therefore, we further propose to use service differentiation via the above latency and reliability improvement techniques. By applying these techniques to inelastic services, it gives an improved CPU utilization and energy consumption, while maintaining better reliability and latency. This improves the overall admission probability. Furthermore, admission of services is also dependent on the throughput constraints imposed at each level of the C-RAN. Therefore, we have also considered throughput constraints as an additional parameter that has to be satisfied for a given service to be admitted. The evaluation is performed based on the above intuitive analysis of the problem formulations.

For further elaboration, let us look at each equation separately. Equation (2.49d) is the total CPU capacity constraint of the edge data center. This equation indicates that the total CPU capacity of the edge data center must be higher than the total service workload demand. At a given time this constraint should be checked for an incoming service with a give workload to be admitted. Equation (2.49e) is the overall reliability constraint. This constraint has to be satisfied for a given service to be admitted, assuming other constraints, such as latency constraint, are satisfied. Equation (2.49f) is the end-to-end latency constraint. This equation indicates that the latency incurred at each stage of service's path must be less than the one a service can tolerate. This is also with the assumption that all other constraints are satisfied. Equations (2.49g) to (2.49i) are throughput requirements constraint for microwave backhaul network, fiber backhaul network, and data center interconnecting network, respectively. Similarly assuming the CPU capacity, reliability, and latency constraint, the throughput constraints at each stage of the network at a given time must also be satisfied for a given service to be admitted.

Now let us simplify the equations. Since we would like to admit as much service as possible from all type of arriving services, let us consider the equation involving the maximization of admission probability $Max(Ap_{C-RAN})$. Maximizing admission probability of C-RAN ((Ap_{C-RAN})) means that we would like to achieve the admission probability to be equal to one ($(Ap_{C-RAN}) = 1$). This means

$$\begin{aligned} Ap_{C-RAN} = Pr(\tilde{\tau}_{max} > \tilde{\tau}_{tot})Pr(\tilde{\rho}_{tot} > \tilde{\rho}_{min}) \\ Pr(\lambda_{tot} < CPU_{tot})Pr(Th_{bs} < Lc_{RAN-max}) \\ Pr(Lc_{bh-mw} > Th_{bh-mw})Pr(Lc_{bh-fiber} > Th_{tot}) \\ Pr(Lc_{dc} > \sum_{i=1}^{M}\sum_{j=1}^{N} aTh_{ij}) = 1 \end{aligned} \tag{2.50}$$

which also means maximizing every components of the equations:

$$Pr(\tilde{\tau}_{max} > \tilde{\tau}_{tot}) = 1 \tag{2.51a}$$
$$Pr(\tilde{\rho}_{tot} > \tilde{\rho}_{min}) = 1 \tag{2.51b}$$
$$Pr(\lambda_{tot} < CPU_{tot}) = 1 \tag{2.51c}$$
$$Pr(Th_{bs} < Lc_{RAN-max}) = 1 \tag{2.51d}$$
$$Pr(Lc_{bh-mw} > Th_{bh-mw}) = 1 \tag{2.51e}$$
$$Pr(Lc_{bh-fiber} > Th_{tot}) = 1 \tag{2.51f}$$
$$Pr(Lc_{dc} > \sum_{i=1}^{M}\sum_{j=1}^{N} aTh_{ij}) = 1 \tag{2.51g}$$

These would be possible when we meet the minimum services demands sets of all the services at every component of C-RAN at a given time (or on average). In mathematical terms, this means

$$\tilde{\tau}_{max} = \tilde{\tau}_{tot} \tag{2.52a}$$
$$\tilde{\rho}_{tot} = \tilde{\rho}_{min} \tag{2.52b}$$
$$\lambda_{tot} = CPU_{tot} \tag{2.52c}$$
$$Th_{bs} = Lc_{RAN-max} \tag{2.52d}$$
$$Lc_{bh-mw} = Th_{bh-mw} \tag{2.52e}$$
$$Lc_{bh-fiber} = Th_{tot} \tag{2.52f}$$
$$Lc_{dc} = \sum_{i=1}^{M}\sum_{j=1}^{N} aTh_{ij} \tag{2.52g}$$

This is a theoretical minimum design criterion for C-RAN system. However, meeting these criterion for all type of services with stringent requirements are very challenging. Therefore, it is necessary to maximize the chance of admitting all type of services by separately looking at some of the constraints. This could be possible by improving C-RAN system's end-to-end reliability and delay constraints, while considering efficient resource utilization and energy consumption.

2.3.4 System Model Formulation with Service Differentiation

This section presents our reformulation of the overall C-RAN system considering service differentiation: backup VNFs to improve VNFs' reliability, and CPU over-provisioning to reduce processing and VNFs' queuing delay. However, providing backup VNFs and CPU over-provisioning are costly leading to a decrease in CPU usage efficiency and an increase in energy consumption. These have a direct implication on CaPex, OpEx and carbon footprint that operators strive to minimize.

Improving VNFs Reliability by Using Backup VNFs for Critical Services

There are various reasons for VNFs failure such as software bugs or physical server restarting or even failure of physical machine[178, 235]. In [220], the authors suggested to use backup VNFs to improve the reliability of a service function chain. Their proposed solution is to increase reliability through redundancy of VNF instances. VNF instances are instantiated as backup VNFs to migrate services in case of active VNFs failures.

Therefore, the reliability of SFC incorporating backup VNFs is defined as [220]

$$\tilde{\rho}_{vnf-with-backup} = \Pi_{i=1}^{N}(1 - \Pi_{i=1}^{N}[1 - \tilde{\rho}_{vnf})] \tag{2.53}$$

$$\tilde{\rho}_{Edc} = [(\Pi_{i=1}^{N} Pr_{link})(\Pi_{i=1}^{K} Pr_{phs})(\Pi_{i=1}^{N} \tilde{\rho}_{vnf-with-backup}] \tag{2.54}$$

Using these techniques, maximizing reliability increases the total number of idle VNFs. Idle VNFs are backup VNFs, which are waiting to replace working VNFs in case of failure or waiting to be scheduled in case of sudden and unexpected workload spikes. However, this requires additional physical servers to be activated. This increases the total cost due to power consumption and computing resource utilization. Therefore, we propose to use backup VNFs for those services that requires ultra reliability. In other words, we only use this technique for Type 1 services. We reserve some number of VNFs considering the required SFC for Type 1 services. Our assumption here is that by providing the required amount of resources as a backup VNF, we increase the reliability of the VNFs to meet Type 1 services' reliability constraint. Nevertheless, that may not be required for Type 2 services. In fact, Type 2 services have softer and tolerable reliability constraint, and an increase in computing resources may not be necessary. Moreover, it could be invaluable and costly for service Type 3 with much relaxed reliability constraint. Therefore, it is necessary to differentiate services to apply backup VNF techniques in VNF placements and service chaining. In doing so, we minimize the expected increase in resource utilization, and energy consumption, while maximizing service admission.

Allocating More Processing Resources to Reduce Edge Data Center Processing Latency for Critical Services

In case of flat scheduling (without service differentiation), the end-to-end latency incurred by the service could be decreased by introducing a CPU over-provisioning technique [10]. Nevertheless, using this method consumes more resources. Considering various services' latency constraint, it is inefficient to dedicate more resources for those services that do not have critical and strict constraints. Therefore, by treating services according to their requirements, we could utilize the resources efficiently.

For service Type 1, the approach is to give as much resource as possible to reduce the processing latency. For the processing latency, we need to make the best possible resource provisioning for VNFs to meet the latency constraint. For service Type 2, the underlining idea is to give minimum resource as possible to meet the latency requirements. This could be an optional decision for service providers to consider making the cost within the tolerable range. In doing so, users of such type could have affordable and flexible pricing. Here, cost is in terms of edge data center's computing resources and energy consumption. For service Type 3, the primary idea is to perform tasks without too much delay or within the tolerable deadline.

$$U_{cpu-vnf} = af(CPU_{vnf}) + b \tag{2.55}$$

where

$$a = \frac{D_{max} \times D_{min}}{\Phi_{min} \times \Phi_{max}} \tag{2.56}$$

$$b = \frac{D_{max} \times \Phi_{min} \times D_{min} \times \Phi_{max}}{\Phi_{min} \times \Phi_{max}} \tag{2.57}$$

In general, applying both techniques, to reduce processing delay and increase reliability, change equation (2.11) and (3.17), respectively. These also change the overall formulation.

Energy Efficiency Calculation

Equation (2.58) [125] is used to calculate the energy efficiency of the above techniques. Using this, the techniques are compared in terms of energy cost. Energy cost has an implication on the operating expense.

$$EE = \frac{Th_{tot}}{P_{tot}} \tag{2.58}$$

It should be noted that the above energy efficiency definition does not fully reflect the energy consumption in comparison to the QoS improvement, such as reliability and latency. For example, we have suggested to use backup VNFs and more CPU resources, to reduce reliability and latency respectively. This leads to higher service admission, specially for Type 1 services. The increase in throughput could not fully incorporate the QoS improvement of the C-RAN system. Therefore, we suggested to develop a more comprehensive measurement technique of energy efficiency, that incorporate QoS improvement as future work.

2.4 Performance Evaluation of the VNF Placement Mathematical Model

In this section, we evaluate the performance of our mathematical model and our proposed VNF resource allocation, based on service differentiation technique. The performance is evaluated in an urban scenario, for the city of Manchester. Most of the parameters are taken form Lu and Di Renzo [161]. An edge data center is assumed to be located in the central offices of an operator. BSs density is assumed to be 37 BS/A, where $A = 1.8$ km^2. Considering a square side of 15 km for the city center, we have about 125 areas, which contains a total of 4625 BSs. For 5G RAN, we

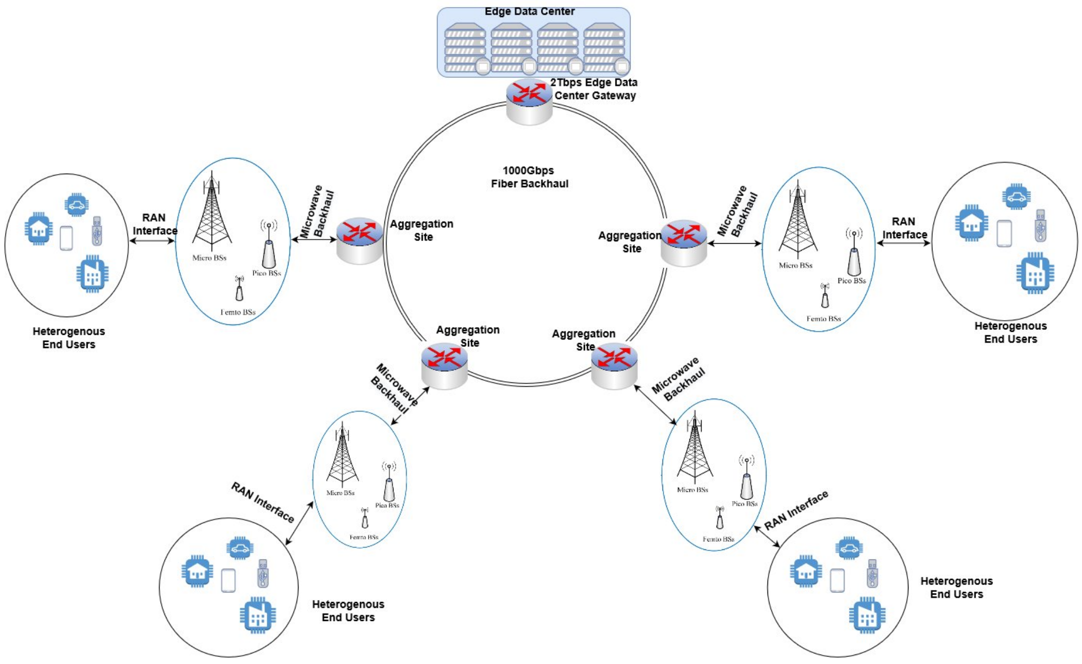


FIGURE 2.4: Sample C-RAN Infrastructure Deployed for Performance Evaluation Scenario

have assumed a massive MIMO with multi-connectivity and interference management and service optimized re-transmission mechanisms to achieve the throughput, reliability and latency requirements of 5G services[213, 193].

We have considered two stage backhaul via MW and fiber links, with indicated parameters as shown in Table 2.2 [126, 178, 235, 213]. We considered simplified arrangement of the C-RAN as can be seen in Figure 2.4. It has a ring fiber network interconnecting all aggregation points to the edge data center. It is assumed that the traffic from all cellular network coverage is aggregated from left and right side of the edge data center through the fiber links. Both links are assumed to have 500 Gbps with the legacy metro link capacity and 1000 Gbps with our proposed link capacity, see Table 2.2. A 54 GOPS processing capacity edge data center is considered for the standard 5G C-RAN [91]. A number of servers are allocated to host various type VNFs to be orchestrated to create the required SFC. A chain of VNFs is assumed to be an ordered subset of queues Q, generated by an ordering function. The SFC ordering constraints required to be applied on the incoming service is a nonlinear complex problem. Instead of solving this problem, we considered a fixed number of VNFs ordering, but flexible VNFs' resource allocation technique. We set few VNFs and some of them working in parallel. The VNFs allocate resources flexibly, being expanded and contracted, in order to perform the required function[10]. Therefore, the problem of VNF placement is considered as instantiating few numbers of VNFs and flexibly allocating required CPU resource, instead of instantiating a number of VNFs with fixed size. This enables to adjust VNFs' resource requirements to efficient and flexible map according to the demands. In other words, in flexible resource allocation, we will not have oversized VNFs, which take extra computing resources. This also provides latency reduction in two ways. First, it gives enough resources to perform the computation as fast as possible, as discussed in the above sections. Second, it avoids adding extra links between the VNFs, as it expands to suit the needs. The reduction is due to the strict resource allocation approach, applied by the

Service type	Reliability	Latency	Throughput	Workload percentage
Type 1	98-100	0.5-50ms	100-200Mbs	15
Type 2	95-98	50-150ms	1-10Mbs	70
Type 3	90	150-650ms	1Mbs or less	15

TABLE 2.1: The three type of services and their requirements

TABLE 2.2: Different Components of C-RAN and Their Constraints

C-RAN components	Latency(Ms)	Reliability(Prct)	Link Capacity(Gbps)	Processing Capacity(GOPs)
RAN	1	≈ 100	0.5/1	N/A
MW-BH	0.65	99.3	1/10	N/A
Fiber-BH	1	≈ 100	500/1000	N/A
EdgeDC-Ntk	0.02	≈ 100	500/2000	N/A
$Edge_{DC-Proc} + Qu_{Delay}$	$= 10/4 + 4/2$	95/98.8	N/A	54/80

model.

We defined the end-to-end latency in the SFC as the sum of delays introduced by each series of VNFs. So, the latency is dependent on the branch that introduces longest delay. Therefore, the processing delay of each VNF is set in the range of 4 ms to 10 ms, depending on the workload L_{ij}, on the allocated CPU and on the VNF processing and waiting (queuing) delay [99, 310]. Processing queuing delay is the waiting time in the queue before being assigned to a given VNF. Transmission delay and network queuing delay of the edge data center network is $\sigma = 0.01$ ms [310]. The reliability of Physical Machine (PM) nodes and VNFs sequence are calculated to be 0.978 and 0.956 respectively. This is the current value achieved by the standard 5G. See Table 2.2 and Subsection 2.4.1 for the values, calculated after applying the backup VNFs.

The proposal is to differentiate traffic in allocating VNF sequences (SFCs). Thus, the incoming 5G traffic is composed of three types of services, as defined in Section 2.3.1. The proportionality equation below is given by

$$\begin{aligned} Srv_{tot} &= \sum_{i=1}^{N_{t1}} \sum_{i=1}^{M_{t1}} Srv_{t1} + \sum_{i=1}^{N_{t2}} \sum_{i=1}^{M_{t2}} Srv_{t2} + \sum_{i=1}^{N_{t3}} \sum_{i=1}^{M_{t3}} Srv_{t3} \\ &= \alpha_1 Srv_{tot} + \alpha_2 Srv_{tot} + \alpha_3 Srv_{tot} \end{aligned} \tag{2.59}$$

where $\alpha_1 Srv_{tot}$ is referred to Type 1 services, $\alpha_2 Srv_{tot}$ is referred to Type 2 services and $\alpha_3 Srv_{tot}$ is referred to Type 3 services.

Since there is no representative data for 5G traffic, the fraction of service composition is assumed to be 15:70:15, in the evaluation. Please refer to Table 4.1 for the service ratio and measurement values used in the evaluation.

2.4.1 Reliability and Latency Constraint Analysis

In this section, we focus on calculating and analyzing different scenarios using the provided performance evaluation parameters. In other words, we would like to

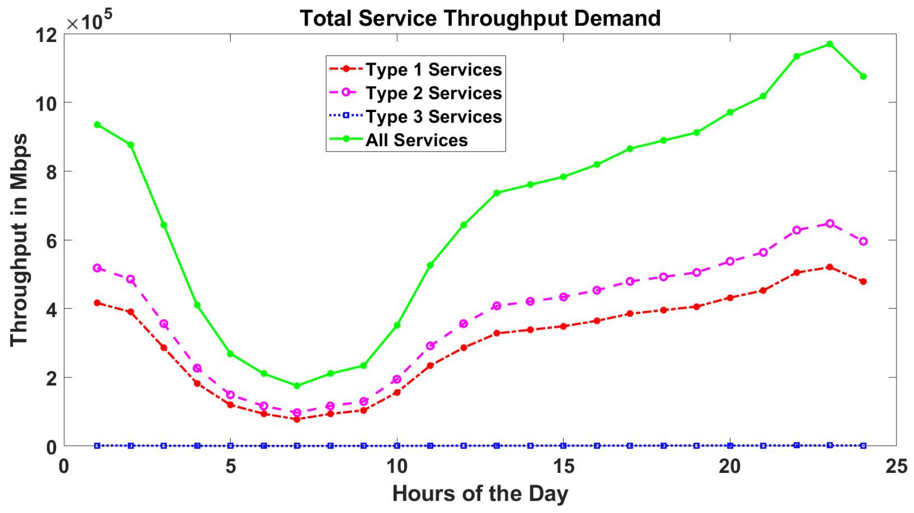


FIGURE 2.5: Total Service Throughput Demand

evaluate the end-to-end reliability and latency imposed by the C-RAN system. End-to-end reliability is calculated considering all components of the C-RAN system, which consists of RAN, microwave backhaul, fiber backhaul, and edge data center (data center network, physical servers hosting the VNFs and the VNFs sequence). The values are provided in Table 2.2 [178, 235, 213]. For microwave backhaul, we have $MTBF_{ntk-bh} = 2029$ and $MTTR_{ntk-bh} = 13$, using reliability equations (3.15), (3.16) and (3.17).

Similarly, we have calculated the reliability of RAN, fiber backaul and data center network and provided the result in Table 2.2. Moreover, we considered the edge data center VNFs sequence reliability in two ways, with backup VNFs and without backup VNFs. Thus, considering the value of $MTBF_{PM} = 5446$, and $MTTR_{PM} = 60$ for both physical and virtual machine, we will have $\tilde{\rho} = 0.978$ without backup VNFs sequence. Therefore, the overall end-to-end reliability becomes $\tilde{\rho}_{tot} = 0.95$ for standard 5G. On the other hand, when we apply backup VNF in the VNF sequence, the overall end-to-end reliability changes, and it is calculated using equation (2.53). The result improves the end-to-end reliability, which becomes $\tilde{\rho}_{tot} = 0.988$.

End-to-end latency, without applying CPU over-provisioning, is calculated as the sum of delays imposed by each components of the C-RAN. This is calculated using value from Table 2.2 as

$D_{end-to-end} = D_{ran} + D_{mw-bh} + D_{fiber-bh} + D_{DC-ntk} + D_{Proc} + D_{Queue} = 1ms + 0.65ms + 1ms + 0.02ms + 10 + 4 = 16.67ms \approx 17$.

After applying CPU over-provisioning to reduce the VNFs processing delay, the end-to-end delay is recomputed as

$D_{end-to-end} = D_{ran} + D_{mw-bh} + D_{fiber-bh} + D_{DC-ntk} + D_{Proc} + D_{Queue} = 1ms + 0.65ms + 1ms + 0.02ms + 4 + 2 = 8.67ms \approx 9$

2.4.2 Throughput Constraint Analysis

Figure 2.5 shows the distribution of throughput demand of arriving services in an edge data center within 24hrs. Using this, we have calculated the required network resources, considering all the three-tier users: femto, picco, and micro cells. For each part of the C-RAN components, we have analyzed and calculated the required network link capacity.

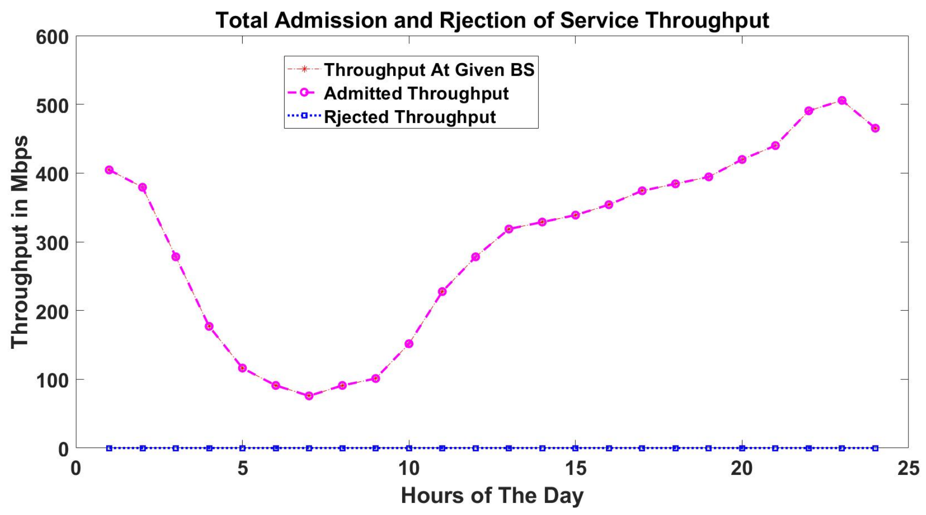


FIGURE 2.6: Admission and Rejection of Service Throughput Due to RAN Constraint

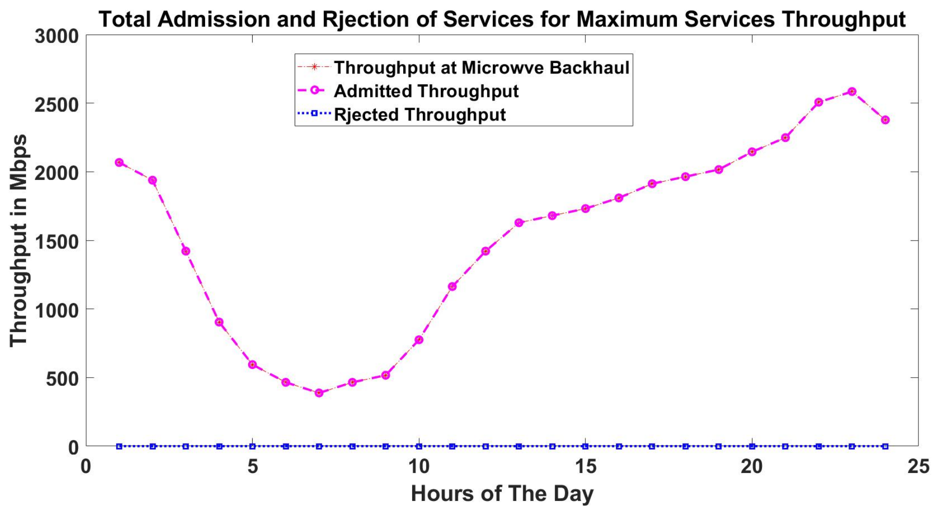


FIGURE 2.7: Admission and Rejection of Service Throughput at Microwave Backhaul

Figure 2.6 shows the admission and rejection of services due to RAN capacity constraints. As it can be seen from the figure, all the services are accepted. It is because the average throughput for RAN is less than the total link capacity which we assumed to be 1 Gbps.

The first stage of the C-RAN backhaul is the microwave backhaul, which imposes link capacity constraints of 10 Gbps to aggregated BS sites' service traffic. Figure 2.7 shows the admission and rejection of service throughput at microwave backhaul. As it can be observed from the graph, the microwave backahul, with 10 Gbps, is able to admit all the services. The second stage of the backhaul aggregation is the fiber backhaul with a link capacity of 500 Gbps in the standard 5G. Based on this link capacity values, we have calculated the service admission and rejection. As it can be seen from Figure 2.8, there is a significant amount of service throughput rejection due to the fiber link capacity constraint. This is true for services in both cases with average throughput and maximum throughput. Therefore, we suggested an increase of the link capacity to at least 1 Tbps in design of fiber backhaul network to admit more

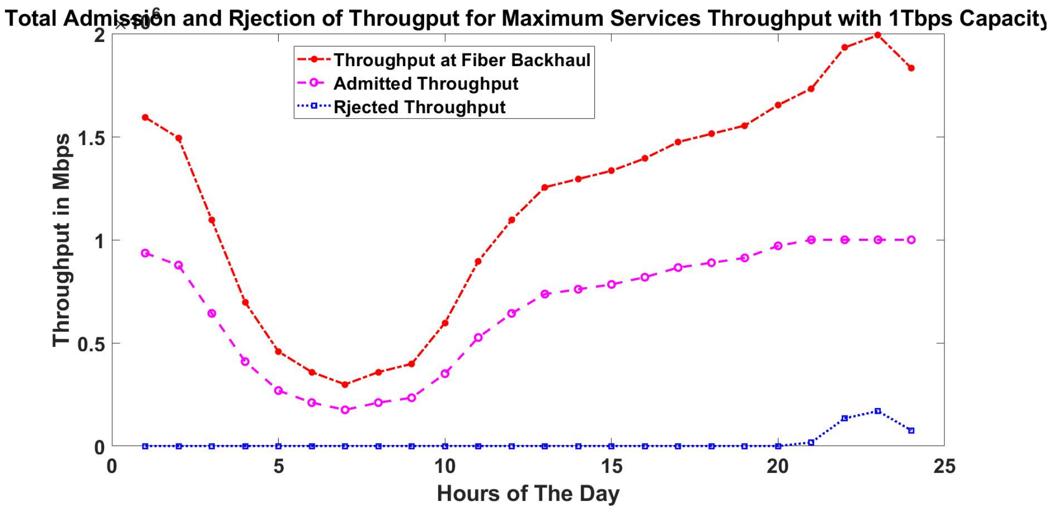


FIGURE 2.8: Admission and Rejection of Services Throughput at Fiber Backhaul

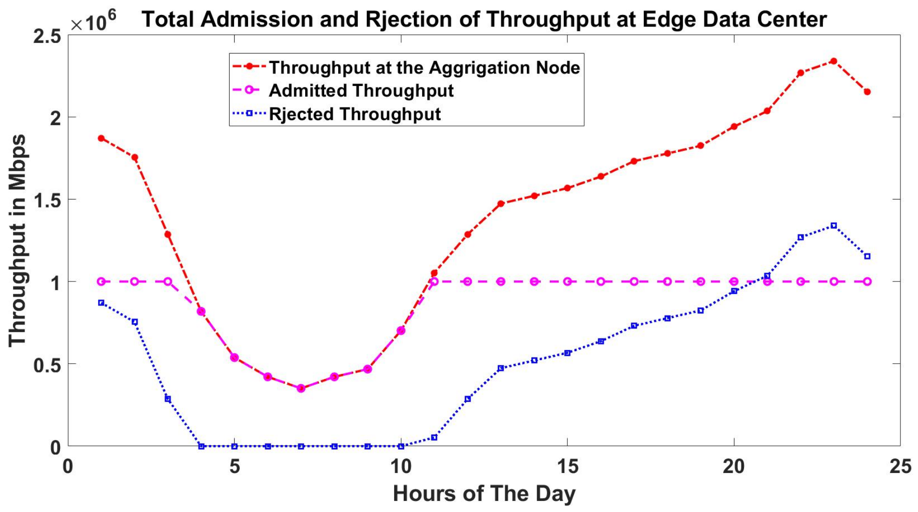


FIGURE 2.9: Admission and Rejection of Services Throughput at the Edge Data Center Network 1 Tbps Aggregation Links

services. The edge data center network is the point where all the aggregated traffic pass through and distributed for service scheduling and processing to the interconnected servers in the data center. The main challenge of edge data center network is the aggregation point congestion. As it can be seen from the Figure 2.9 with the current aggregation link capacity of 500 Gbps at the edge data center, there is heavy loss of service throughput due to aggregation node congestion.

Therefore, we suggested to have a network aggregation link capacity of at least 2 Tbps in the edge data center network design. This alleviates the congestion problem to admit more service throughput.

Edge Data Center Computational Capacity Constraint Analysis

Figure 2.10 below shows arriving service workload demand distribution in an edge data center in 24 hrs. Considering a 54 GOPS of computing resource for the standard

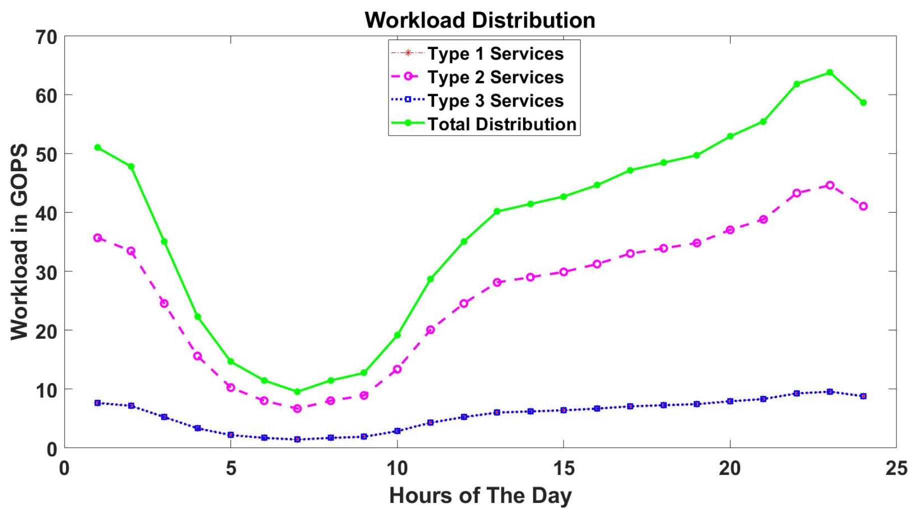


FIGURE 2.10: Arriving Workload Traffic Distribution

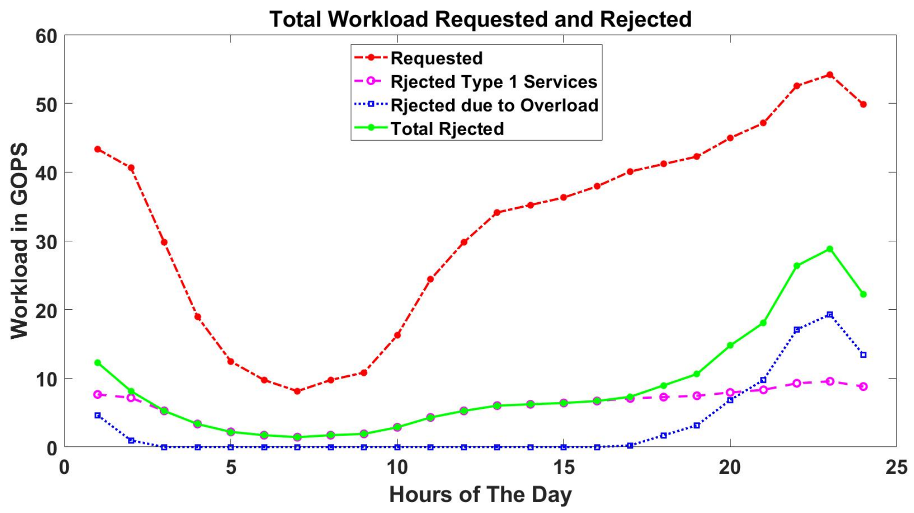


FIGURE 2.11: Workload Requested and Rejected

5G data center, we calculated the service rejection as depicted in Figure 2.12. The figure shows the total requested and total rejected service workload. The total rejected service workload is a composition of workload rejection due to: standard C-RAN inability to meet 5G latency and reliability requirements constraints for service Type 1 and computing resource constraints for service Type 2 and 3.

Figure 2.12 shows the total workload requested and rejected. The figure indicates that there is an amount of service workload rejection due to computing capacity constraints. Moreover, there is a complete rejection of Type 1 services because of service constraints. Thus, we suggested to apply the above mentioned techniques to increase reliability and reduce latency.

Therefore, to meet the reliability constraints, we use backup VNFs. However, the resource demand becomes doubled resulting in more service rejection due to computing resource constraints. Therefore, figure 2.12 shows the service requested and rejected due to resource constraints in the edge data center with backup VNFs. The figure shows the rejected workload nearly doubled while the total computing capacity is fixed. Moreover, we would like to provide more computing resources (over-provisioning) for each VNFs to reduce the service processing latency as much

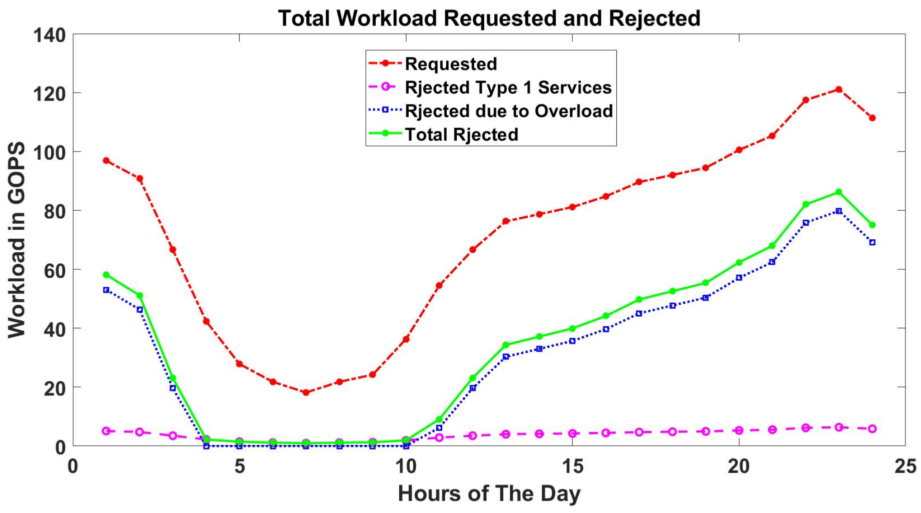


FIGURE 2.12: Total Workload Requested and Rejected using Backup VNFs and CPU Over-provisioning for Reliability and Latency Improvements, Respectively

as possible to admitted more Type 1 services. However, this also takes more computing resources, resulting in greater service rejection (see Figure 2.12). To reduce service rejection without increasing computing resources, we use backup VNFs and CPU over-provisioning only for Type 1 services. This reduces the total required CPU resources by 85%. Note that the result is based 15:70:15 service composition, see Figure 2.13. This may vary depending on network coverage areas and operators and so on. Using the equation in Section 2.3.4, we calculate the admission probability for different cases, see Figure 2.15. The result suggests that applying service differentiation provides greater admission of services without increasing computing resources. However, there is still significant amount of service rejection due to overload. Therefore, we suggest the edge data center to have minimum of 80 GOPS for acceptable admission of services' workload.

Figure 2.16 shows, for a given service workload, the comparison of total resource consumption among three different service function chaining and placement techniques. The service workload demand is depicted in red color. The lines in blue, purple and green are total resource consumption comparison among fixed offline, fixed online and flexible offline techniques for comparison. Fixed offline strategy follows allocating fixed number of VNFs with fixed amount of VNF resources in the SFC. For fixed online each, VNFs are also allocated fixed resources but the number and ordering of VNFs is determined online. Th flexible offline that we adopted allocates fixed number of VNFs in the SFC but the VNFs' resource is dynamically allocated as per the demand. The fixed offline and flexible offline resource allocation techniques are formulated based on Mixed Integer Linear Program (MILP)[10] and fixed online is based on Mixed Integer Program (MIP) [202]. Fixed allocation techniques consume more than the flexible one that we have proposed. This is because of multiple factors such as virtualization overhead due to oversized VNF instantiation, online computation for placement and ordering solution, and communication overhead [202]. Fixed online resource allocation provides the most CPU resource consumption followed by the fixed offline. The gain is dependent on the amount of workload at a give time. The relative gain of the flexible method over fixed offline technique is 9.9% and over fixed online is 15.5%.

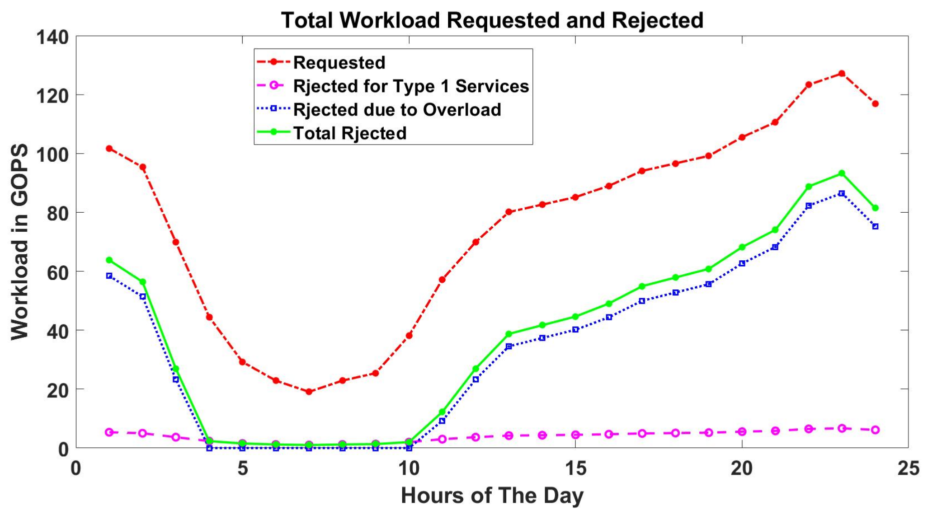


FIGURE 2.13: Total Workload Requested and Rejected using Backup VNFs and CPU Over-provisioning for Reliability and Latency Improvements Respectively with Service Differentiation

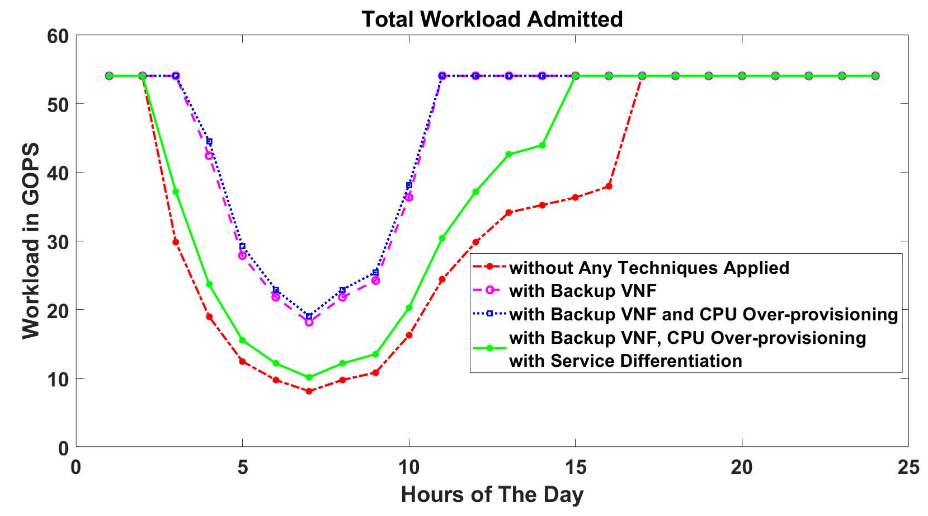


FIGURE 2.14: Total Workload Admitted

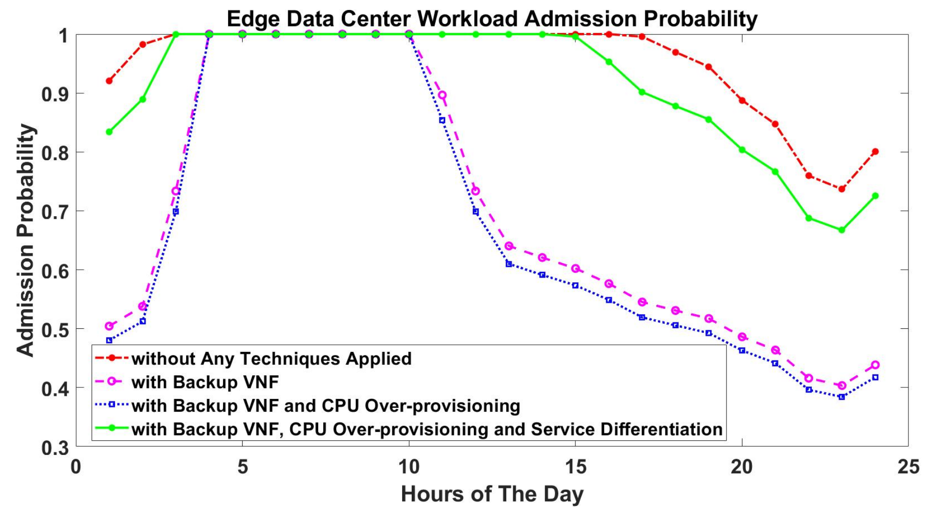


FIGURE 2.15: Edge Data Center Workload Admission Probability

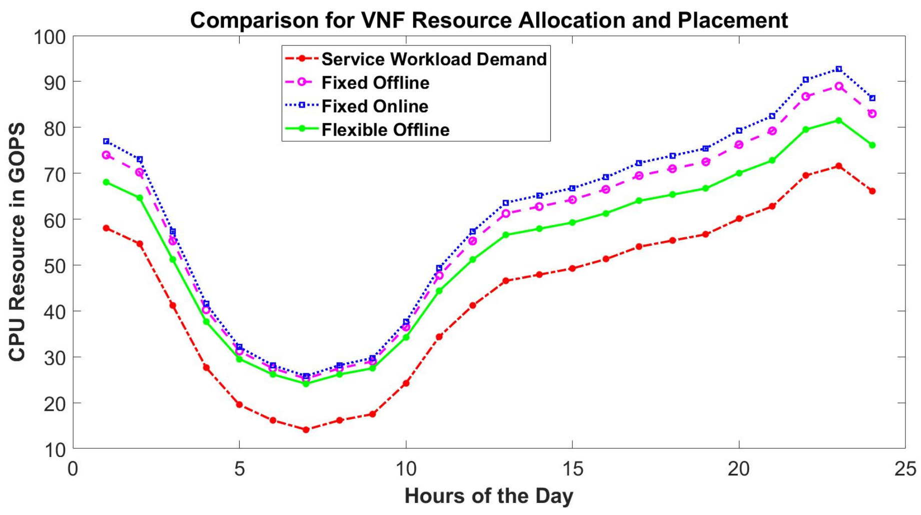


FIGURE 2.16: Used CPU Resources Comparison for the Given Service Workload.

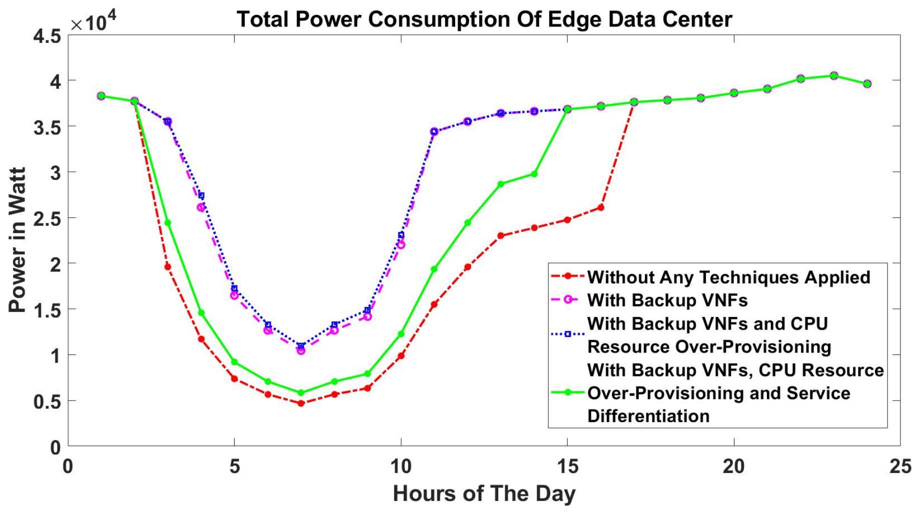


FIGURE 2.17: Total Power Consumption of Edge Data Center in Different Cases

2.4.3 Power Consumption and Energy Efficiency Analysis

Finally, we have calculated the power consumption for the two applied methods to improve reliability, and latency. The result is plotted in Figure 2.17.

Figure 2.18 shows the energy efficiency for several cases. In general, the result suggests that the applied service differentiation technique saves more energy than without applying it. However, because of the backup VNFs and CPU overprovisioning that are applied to improve the SFCs' reliability and processing latency, our method consumes additional energy to serve a given service. Nevertheless, it should be noted that this is the cost of attempting to admit Type 1 services meeting its constraints.

The overall performance for our proposed model has suggested backhaul link capacity of 1 Tbps, edge data center aggregation link capacity of 2 Tbps and 80 GOPS CPU resource capacity at edge data center. That is plotted in Figure 2.19. It is drawn

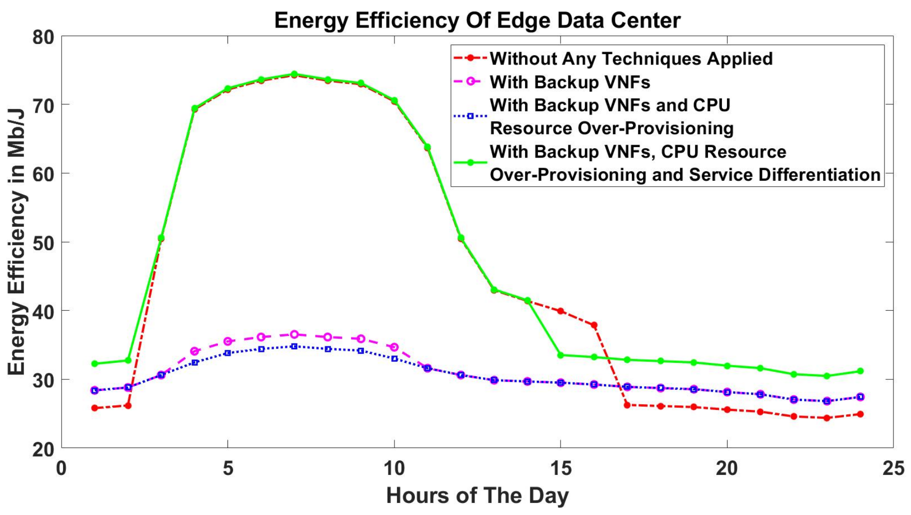


FIGURE 2.18: Energy Efficiency of Edge Data Center

as a radar plot considering the six 5G representative requirements: latency, reliability, user experienced throughput, CPU usage, energy efficiency, and admission probability. The figure summarizes the overall comparison among the existing 4G/LTE standard capability, our proposed method and the standard 5G performance requirements. In terms of the six 5G constraints, the existing standard 4G performance is far from fulfilling most of 5G requirement, which can be seen from the figure. However, the resulting evaluation with the proposed techniques shows a significant improvements over the standard 4G capability. As it can be observed from the plot, the performance of our proposed method is very close to the 5G requirements. This is represented in the figure by the designated area.

2.5 A Translator as Virtual Network Function for Network Level Interoperability of Different IoT Technologies

Now let us have a deture and provide an example of how to develop and network function as VNF. Here we develop a translator as VNF for network level interoperability of different IoT technologies. The Internet of Things (IoT) market is rapidly growing revolutionizing and impacting various sectors such as healthcare, agriculture, energy harvesting, transportation, etc. This is mainly because of the possibility of pervasive connectivity among machines, objects, humans, and virtually anything. Currently, there are billions of devices connected and hundreds of platforms integrated. The technologies for IoT connectivity should enable the interconnection of billions of heterogeneous devices with stringent and diverse requirements. To address these requirements, different IoT technologies have been introduced. They also use different protocols. They are designed differently according to the targeted problem they address. Each IoT technology is suitable to address a particular challenge. For example, unlicensed (LoRa) is better suited for applications requiring a long battery lifetime, low capacity, and cost. Whereas, licensed NarrowBand IoT (NBIoT) suitable for applications requiring better Quality of Service (QoS), latency, reliability, and range.

As mentioned above, to address specific requirements, the different IoT technologies are developed with different technical principles. For example, each may use different techniques for signaling, coding, communication protocols, and data

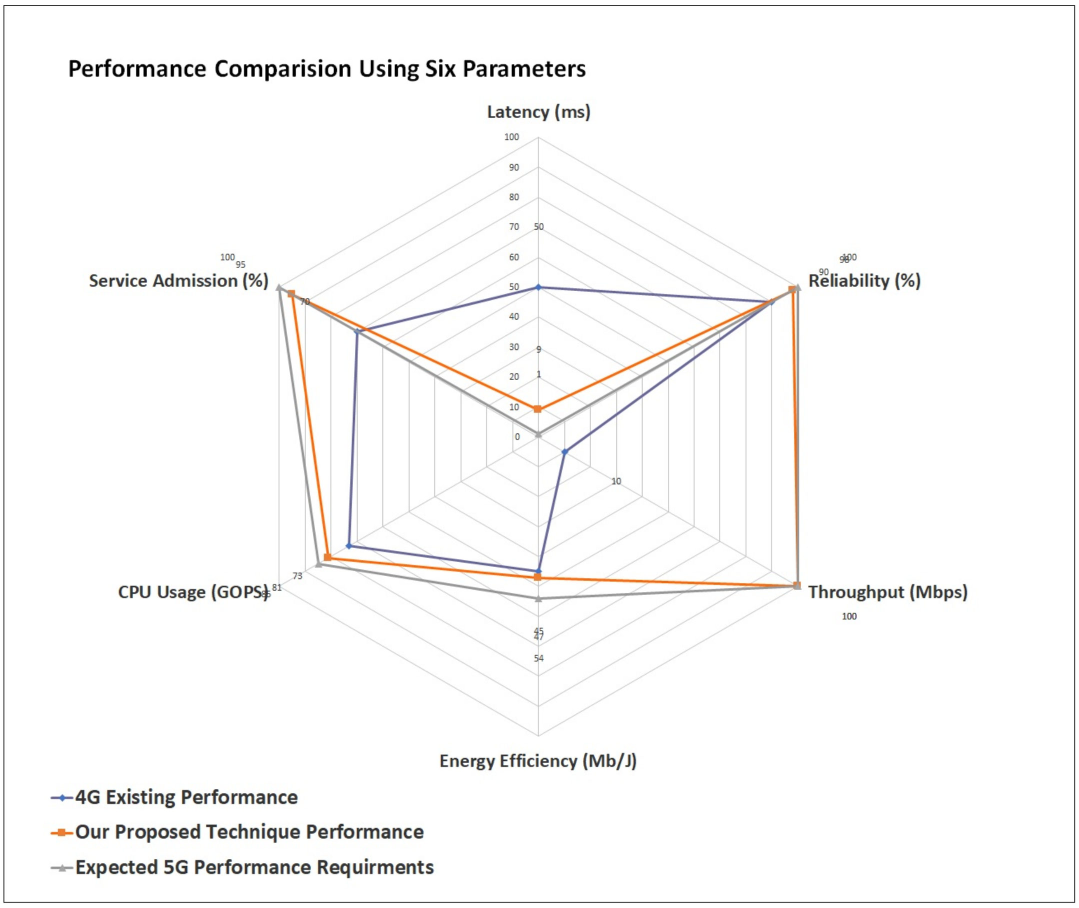


FIGURE 2.19: Overall performance of the proposed model in comparison with the existing 4G performance and the expected 5G requirements

formatting. However, even if these techniques enable them to address a particular application requirement, they can become a barrier to interoperability. This interoperability problem, for different IoT technologies, is prohibiting the full utilization of IoT and its potential applications. For example, if we consider IoT for space application, a single martial or space mission needs to gather multiple pieces of information about the target planet. Hence, the mission may employ multiple IoT devices to be deployed on the martial surface. For this, it may require having various types of possible connection interfaces among units. Whereas, for environmental monitoring requirement, it could be satisfied with unlicensed LoRa-based IoT-devices for environmental measurement parameters, such as temperature, humidity, soil content, and so on. The processing of the collected data from different IoT technologies would reduce the size of the integrated data for efficient transmission. Interoperability enables this possibility.

The benefit of interoperability is even more pronounced on earth as we have numerous IoT technologies being implemented in various locations. Mostly this is applicable in overlapping network coverage using different IoT connectivity technologies. Different IoT technologies may exist in a given geographic location. The overlapping and coexistence could be necessary to provide different types of connectives for various applications. However, the coexistence could be utilized for further exploitation of the IoT resources for more applications or more efficient utilization in the same applications. Therefore, interoperability between each technology is crucial in the progress and universal adaptation with efficiency, and cost-effectiveness of IoT devices and connectivity technologies. This could also be for energy efficiency, resource consumption, QoS, and so on to be exploited in time and space.

There are various levels of interoperability, such as device-level, network-level, syntactic level, semantic level, cross-platform, cross-domain interoperability. For full interoperability between IoT technologies, it is necessary to speak at each level. However, depending on the application it could be sufficient to have one or two levels of interoperability. There are few works on network-level interoperability. Therefore, this work focuses on network-level interoperability. In particular, this work aimed to define a unique virtualized access point (VAP) capable of connecting devices belonging to different technologies. Network function virtualization and software-defined networking technologies are considered to improve the developed translator deployment and resource usage for dynamic, flexible, and cost-effective deployment of the protocol translator.

2.5.1 Overview of IoT Technology Interoperability

Let us provide a brief discussion of the various levels of interoperability, along with a possible structural representation. The ability of two different IoT systems to interoperate can be presented using different types of layered models. An example could be the six-level structure;

- No connection: which means no interoperability between the IoT connectivity systems
- Technical: which is basic and network connectivity
- Syntactical: which is data exchange interoperability
- Semantic: which is understanding the meaning of the data
- Pragmatic/dynamic: which is the applicability of the information

- Conceptual: which is shared view of the world

IoT interoperability can also be seen from different perspectives, such as device interoperability, networking interoperability, syntactic interoperability, semantic interoperability, and platform interoperability. This can be used to study the problem at different abstract levels.

Since our proposed solution focuses on network-level interoperability, particular attention is devoted to this. In the literature, different types of layered models have been provided to split the interoperability problem into more simple sub-problems. Among them, [196] and [194] to be the most general ones. The former [196] provides a well-structured representation by using a hierarchical arrangement, where details increase as we go up the hierarchy. In this paper, we adopt the later [194] because we believe it is better to combine completeness and simplicity. In the following the structure provided by [194] will be used to define interoperability at each level. [196] will also be used as a reference to provide a better description. Now, let us discuss further details of the six-layer model which is defined in the above section.

- **Device level interoperability** is enabling the integration and interoperability of heterogeneous IoT devices with various communication protocols and standards supported by the heterogeneous IoT devices. Device-level interoperability is concerned with the exchange of information between heterogeneous devices, heterogeneous communication protocols, and the ability to integrate new devices into any IoT platform[31, 111]. Device-level different IoT platforms can be characterized by different communication technologies possibly operating at different frequencies. Since direct interoperability cannot yet be achieved at this level, gateways are often used as an intermediary in communications.

- **Syntactical level interoperability** is interoperation of the format as well as the data structure used in any exchanged information or service between heterogeneous IoT connectivity system devices. Syntactic level considers the possibly different "rules" used to encode and decode messages. Semantic level syntactic level interoperability provides a way to correctly encode and decode data. Semantic interoperability aims instead at defining a way to correctly understand data that may be represented in different ways. For instance *"temperature"* and *"temperatura"* can be used as different words for the same concept and *"Fahrenheit"* and *"Celsius"* as different units for the same value. On one hand, standard semantics can be used to represent data, on the other hand, machine-readable dictionaries can be used to translate from one semantic to another. Different IoT technology will have different structures for data representation. The Web of Things (WoT) as been exploited at this level since the web supports different representations such as XML and JSON. According to some schema, to expos some structure, An interface needs to be defined for each resource. Examples of syntactical level interoperability are WSDL and REST APIs[196, 31, 111].

- **Semantic level interoperability** is defined as "enabling different agents, services, and applications to exchange information, data and knowledge in a meaningful way, on and off the Web"[286].

- **Platform level interoperability** is the interoperation of different platforms. In IoT, this arises because of the existence of various operating systems, programming languages, data structures, architectures, and device and data access mechanisms[196]. The cross-platform level provides interoperability for different platforms belonging to the same field of interest, e.g. an app able to control the heating system both at home and in the office.

- **Network level interoperability** is mechanisms to enable seamless message exchange between systems through different networks (networks of networks) for end-to-end communication. The network-level key point for this level is to allow end-to-end message exchange when considering devices possibly belonging to different technologies. To make systems interoperable, each system should be able to exchange messages with other systems through various types of networks. Due to the dynamic and heterogeneous network environment in IoT, the network interoperability level should handle issues such as addressing, routing, resource optimization, security, QoS, and mobility support[31, 111].

- **Cross-domain level:** further extends cross-platform interoperability by considering platforms from different domains, e.g. lightning and cooling systems. Usually, if Cross-domain interoperability is provided, so it is also Cross-platform.

Our proposed method relies primarily on network-level interoperability. Moreover, the proposed solution utilizes the concept of SDN, NFV, and SDR. VMs and containers and provides a flexible yet simple deployment of translators. The translators are used to enable the communication between the devices between two different IoT devices for different connectivity technologies.

There are more than 360 different IoT platforms available in the market [74]. Motivated by their primary goal of improving IoT devices' performances and power consumption, vendors have created more diverse structures for their devices thus preventing reciprocal communication and cooperation possibilities. However, as reported in [195], few efforts have been devoted to solving the interoperability problem at the network level. Some of the relevant works are [216] [222] [46] [76]. In [216] a middleware has been used to tackle network-level interoperability in a publish-subscribe fashion. It exploits both constrained application protocol (CoAP) and message queue telemetry transport (MQTT). The middleware is used as a bridge from BLE and 6LoWPAN technologies to the rest of the network. Since both technologies are not able to directly communicate with the rest of the network, some additional modules have been implemented in the middleware. With this type of design, where required components are hardcoded in the middleware, the introduction of new technology in the system is problematic. This is because it requires the user to reach the middleware and implement the new necessary parts on it so that the new technology can be connected to the network.

In [216], the central component chosen is a Raspberry-Pi and other IoT devices are connected to it. Similar work has be conducted in [222]. In [216], a BLE module and relative components has been used. However, in [222] Zigbee has been chosen instead of 6LoWPAN as second technology.

Perhaps, a complete and general solution is described in [46]. In this work, similarly to the above solutions, middleware is used to ensure interoperability among heterogeneous devices. However, in this case, the designed solution exploits the benefits of both cloud and edge middleware deployment to provide improvements in different scenarios. The deployment could be on the cloud for deep analytics. And on an edge data center, near the IoT Gateways, for local analytics to support real-time applications [46]. Data Acquired from heterogeneous devices is stored in an SQLite database using JSON format. Then the data is made available through publish/subscribe fashions using dedicated protocols, such as MQTT and CoAP.

Interesting work has been conducted in [76]. A service-oriented architecture (SOA) is proposed as an alternative to middleware-based solutions. The proposal is based on multi-protocol translations. An intermediate format has been used as a mid-step when translating among two different protocols. The authors have justified such a decision stating that direct protocol-to-protocol translation requires several translators. Whereas, the use of an intermediate global protocol increases the translation time. The translator is implemented in a local cloud. This solution to the interoperability problem is designed to solve interoperability issues at the network level since evaluation has been carried out studying HTTP and CoAP components. However, the idea of protocol translators seems promising.

Virtualization of APs has recently been studied in the literature. In [282] a virtual AP solution for Wi-Fi is designed to improve flexibility and load balance. In the paper, virtual AP is indeed moved between physical APs to provide improved resource balance. Besides being proof sustaining the possibility of implementing virtual APs, this work also provides additional functionalities that could be exploited in our proposal to provide better load sharing.

In general, the problem of interoperability has attracted attention both from the research community and industries. As such several projects have been founded both from worldwide organizations and by the European Commission. Among them, Horizon 2020 European Project INTER-IoT [95] [183] aims to design, implement and test a framework that will allow interoperability among different IoT platforms [183].

Furthermore, the Horizon 2020 European Project INTER-IoT [95] [183], which is probably the closest one to our work, further strengthens the possibility and exigency of works in this newly emerging field. With that being said, we decided to further prove the possibility of creating such a system employing simulation. According to the authors' knowledge, this is the first work studying a solution for network-level interoperability using softwarization technologies such as SDN, SDR, and NFV, where the solution is implemented on a virtualized access point. The translators are designed as VM or Container to be instantiated and placed on demands.

2.5.2 Proposed Network Level Interoperability Architecture

In this section, we present the description of our proposed solution in comparison with existing network-level interoperability architectures. We have identified four main approaches, namely: devices' vendors friendly, standard friendly, cumbersome, and virtualized. Devices' vendors-friendly solutions exploit dedicated access points to connect heterogeneous IoT devices to the internet. In this case, vendors are not required to change their device structure and can stand with their proprietary protocols. This type of solution is the one with fewer interoperability capabilities. Standard-friendly solutions could provide a higher level of interoperability by defining a unique standard, which is widely accepted and used by IoT devices. However, considering the high resource constraint such as power, bandwidth for many IoT applications and devices, interoperability solutions of this kind are not perceived as efficient, at least for the moment. A natural alternative is protocol convergence. Is it possible for IoT to converge on a single shared protocol? This is not unprecedented, the Internet has seen convergence on the Internet protocol (IP). At this time, it is unlikely to see convergence on a single IoT communication protocol. However, there is no well-established reference standard for IoT platform technology [76, 95].

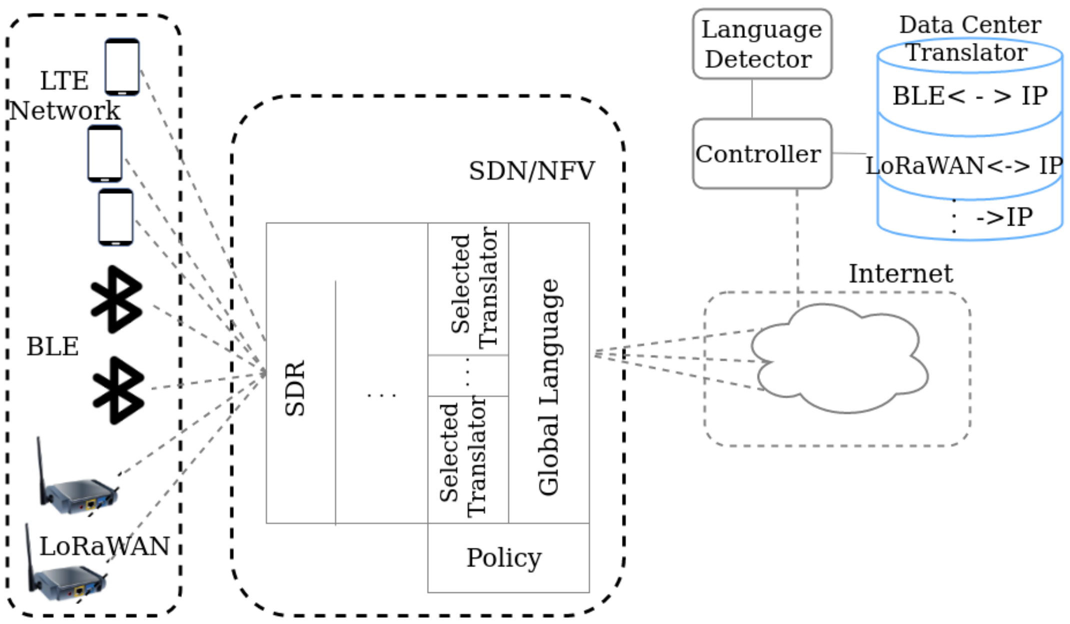


FIGURE 2.20: Proposed Virtualized Solution Structure.

Cumbersome solutions have been introduced as a trade-off between devices' vendors friendly and standard friendly. Solutions of these types have been studied in the literature to obtain a single Access Point to which devices belonging to different technologies can be connected. The word *cumbersome* has been used in this context to highlight low flexibility and costly operations characterizing such types of solutions. Cumbersome-like solutions can deliver a reasonably quick and valid solution to the interoperability problems with limited flexibility for a dynamic network.

IoT technologies are dynamic and heterogeneous. Therefore, more flexible solutions are needed to sustain the increasing number and type of IoT devices. Moreover, a flexible interoperability solution is required to address the dynamic requirements, interaction, and behavior of the various IoT devices. Furthermore, under the current technological trends, softwarization and containerization would provide flexibility in deploying such solutions. Network softwarization paves the way for more dynamic solutions such as the dynamic deployment of a translator based on the current-network interaction and dynamic behavior.

Here we proposes a virtualized solution which is developing a translator to be deployed as a container, such as a docker container. The approach exploits the use of novel technology such as SDN, SDR, and NFV to increase the capabilities of cumbersome-based solutions allowing for improved flexibility. There are other similar works proposed that use SDN, SDR, and NFV technology to tackle the interoperability problem. However, a detailed description of how the joint use of such technologies can be used to solve the network level interoperability problem is unavailable in the literature. Thus in Figure 2.20 we present our proposed architecture. The key components used in our proposed architecture are the virtualized access point, the controller, and network function virtualization. The controller is used to enable seamless routing between the IoT devices that use different IoT connectivity technologies. The NFV orchestrate enables the deployment of a particular translator as virtual components needed by an access point. The key idea of our proposed approach is the development of network-level protocol translators. The translator can

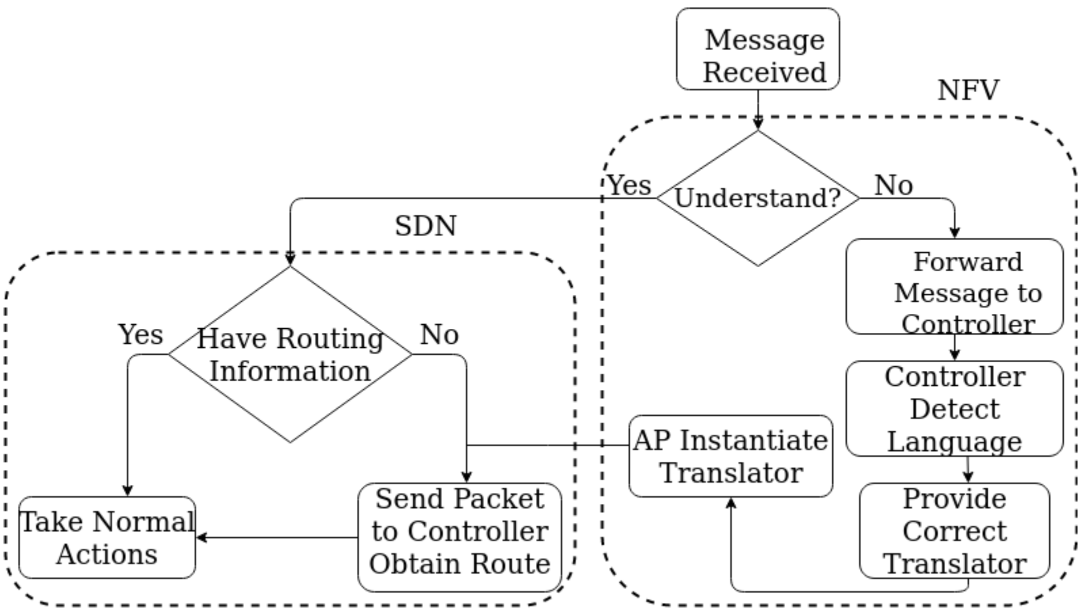

Message Received
NFV
Understand?
Yes
No
SDN
Forward Message to Controller
Have Routing Information
Yes
No
Controller Detect Language
AP Instantiate Translator
Provide Correct Translator
Take Normal Actions
Send Packet to Controller Obtain Route

FIGURE 2.21: Proposed Virtualized Solution Workflow

be used to translate from one device language to another. It provides protocol conversion at the network level. Conversion can be achieved by instantiating dedicated translators in the VAP, whenever needed. The development of this translator as virtual components provides networks with the flexibility that cumbersome solution was lacking. It is also possible to instantiate and deployed the translator as containers. Containers are well known to be more light-way and fast than VMsTo better understand system components and behavior of the system the diagram in Figure 2.21 can be used. Since the translator is provided as virtual machines or containers, it can be easily instantiated and de-allocated. Following the diagram in Figure 2.21 can be observed how the system directly takes routing procedures when the AP can understand the incoming message i.e. if it does have the correct translator. If it does not have the appropriate translator, it has to be obtained first by instantiating the required type of translator as VNF to be placed for translation. This dedicated VNF translator will take care of the required procedures to understand the network-level language used for the packet. And then the virtualized AP is provided with the appropriate translator which will be used to translate the packet language and forward its content towards its final destination. To optimize resource usage, the de-allocation of unused translators will be also provided through the use of internal policies.

The instantiation and placement of a new translator in the VAP is a procedure that took place when a new type of IoT technology is introduced in the network or has been in an idle state for a long time. Therefore, this can be handled using remote components. The language detector module and the DB containing translators can therefore be deployed on the cloud or the edge. It can not be the same for the translation operation. Nowadays IoT scenarios require ever decreasing latency to sustain scenarios like self-driving cars in which networks resilience and low latency are vital requirements. Therefore, a proactive allocation approach could also be implemented once the translator is provided as a container.

Another component that can be deployed directly in the VAP is an SDR module.

This module provides interoperability at the device level. This allows the proposed solution to operating with more IoT technologies, operating at different frequencies. Use of SDR modules in this context has already been studied in literature both for IoT networks [265] and for cellular Networks [311].

Finally, it is noteworthy to highlight the fact that, given the non-triviality of the proposed design, partially due to the ever-increasing number of IoT devices and relative protocols, two main actors could help in realizing the proposed solution. On one hand, vendors could provide translators since they are the ones that better known their protocols and, in this way, they will see their devices immediately connected to every VAP. On the other hand, the community may also be involved to design new and original translators.

The choice of virtual machine deployment and container-based translator deployment is dependent on the required application of the deployed IoT, availability of resources, operator business policy, and other technical and business factors. For example, the two technical aspects in choosing virtual machines over container or vise versa are Data protection and available services. In [224] a description of the dependency by several factors when considering security in VMs and container is provided: *"Comparing container and VM security yields no runaway winner. Much depends on how the containers and VMs are used, and specifically on the architecture of the applications they support."*. Containers are advantageous as they are light-way and fast than VMs. However, the trend adopted by major Infrastructure as a Service (IaaS) is to deploy Containers on Virtual Machines. This gives security and maintenance advantages[64]. However, both technologies represent valid solutions depending on the area of application.

2.5.3 A VNF Based IoT Interoperability Translator Simulation and Performance Evaluation

In this section, we present a simulated implementation of our proposed solution as a proof-of-concept. We used the implementation for performance evaluation of the proposed approach in terms of protocol translation delay in a virtualized environment. The simulation is a basic implementation of the cumbersome solution. It demonstrates the possibility of providing different interfaces for different IoT connectivity technologies. It provides a basic network-level translation mechanism for packets traveling between different IoT connectivity technologies. Various types of interconnecting technologies are deployed to interconnect IoT devices using the protocol translation functions. These translation functions could be deployed as VNF in a standard container or virtualized environment.

The simulation environmental setup consists of a Linux workstation (Lenovo) T480 i7 8550U quad-core with 16 GB of RAM. An ns3 simulation environment hosted on Kubuntu 18.04 VM with 9 GB of RAM and four processors (acceleration VT-x/AMD-V). A flow monitoring module from ns3 is used for sampling and generating the required statistics about packets transmitted through the developed IoT network. However, there is current implementations limitation with the probes and classifiers functions of the flow monitoring modules, these available only for IPv4 and IPv6 [94]. Thus the monitoring tools are used only to track the complete packet flow from the translator and to the IP network. We measured the transmission delay, assuming a similar delay in the transmission of other IoT connectivity networks.

The overall simulation scenario is depicted in Figure 2.22. As it can be seen from the figure, it is organized into three levels. At the top, the BackBoneNode is used as a connection from the local network to the internet or acts as a remote node. The

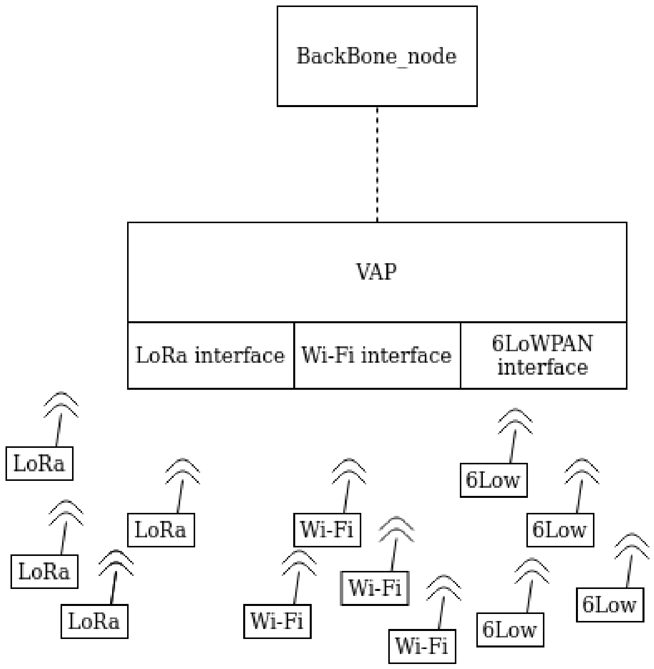


FIGURE 2.22: Implemented Network Topology

middle layer is a virtualized access point (VAP) containing our proposed translator. It is linked to the BackBoneNode using a point-to-point connection. Finally, the third layer contains the connectivity technologies used. Wireless connections are used to connect the end devices with the AP depending on the connectivity technologies considered. These are LoRAWAN, Wi-Fi, and 6LoWPAN. For each technology, four nodes are used as the initial default value but their number can be changed at the run time of the simulation. The simulation is implemented to provide the possible users to change the number of nodes for each technology, the IP version, and to execute the simulation for a portion of the network at a time. Moreover, logging and tracing features can similarly be enabled through the command line at run time.

The NS3 simulator has a *"lorawan"* module which is provided with two classes: *forwarder-helper* and *forwarder*. These can be used to forward the received LoRa packets to another node, by using the provided point-to-point connection. We have exploited and extend these two classes by adding UDP capabilities when forwarding the packet. Using [289] as reference, a lookup table is created in the VAP to store LoRa to IP addresses dependency. Since LoRa devices are not originally designed to work with an IP network, they do not have an IP address.

In the simulation, the associated IP addresses are used in LoRa packet forwarding procedures, while a static node is chosen as the destination. Using the provided logging feature and other tools such as Wireshark and NetAnim, measurements are taken to verify packet reception in the destination node. Wi-Fi and 6LoWPAN implementation follow standard NS3 network implementation using the corresponding modules. Their implementation has been slightly redesigned to better match the coding style used for the LoRa network.

A Python script has been used to generate statistics, assuming no packet priority with the First come First service scheduling technique. Similarly, the time needed to provide the basic translation mechanism has been measured. Using the NS-3 clock

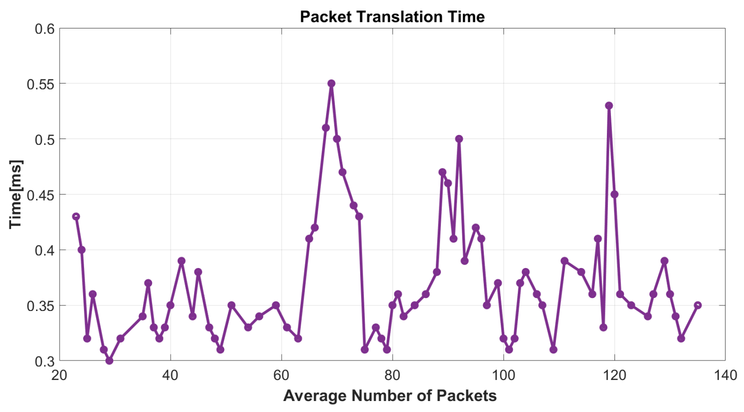


FIGURE 2.23: Translation Delay

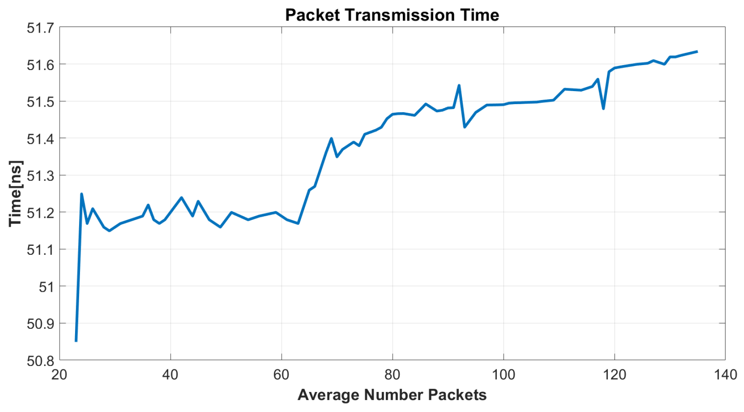


FIGURE 2.24: Translation and Transmission Delay

modules, we measured the departure and arrival time of the packet when traversing through the translator. The measurement is plotted in figure 2.25. Moreover, the translator is expected to be deployed in the container. This will have further virtualization overhead delay[202], adding to the translation delay.

Figure 2.23 shows the time needed to translate an arriving packet from LoRa network to IP network at the network level. As depicted in the figure, the average translation delay is approximately 0.4ms. This is only the time it takes for the translator after receiving a given packet and translator the packet format from source network protocol format to the destination network protocol format.

Figure 2.24 shows the time needed to send a LoRa packet from the LoRa network to the IP network, showing transmission delay incorporating translator delay. It does not Consider the virtualization delay of the translator. It only shows the time needed to send a given packet from one IoT network domain to another IoT network domain using the packet translator. The average delay is slightly below 60 ms. As can be seen from the figure, the time slightly increases when the packet size increases. However, in some cases, the same increment produces a higher variation of time, as can be seen from the left part of the graph. The packet transmission time is coherent with the one obtained in [146]. However, we have used 125kHz instead of 250 kHz

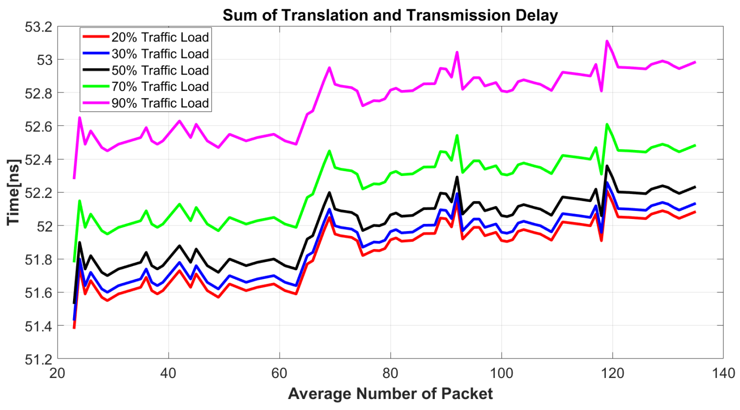


FIGURE 2.25: Transmission Delay at Various Traffic Load Conditions

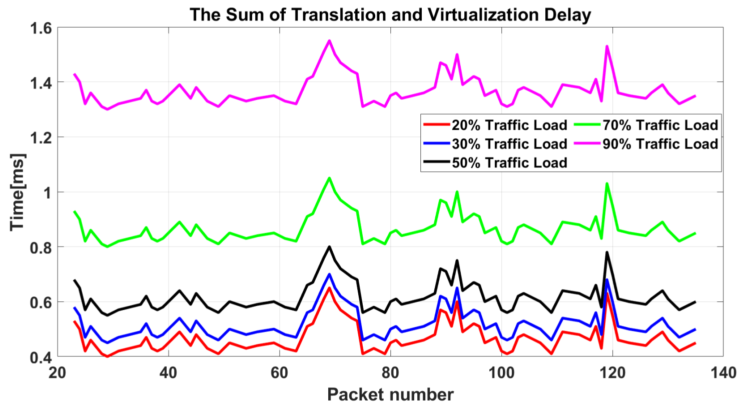


FIGURE 2.26: Transmission Delay Using a Virtualized Translator at Various Traffic Load Conditions

with SF:7. Moreover, a point-to-point channel is used, which introduces a 2ms delay. The translation time may vary if sophisticated translation mechanisms are applied. Nonetheless, we believe that its value will not influence too much the transmission time, especially when considering limitations imposed on some IoT devices. Moreover, a proactive setting of the translator could further reduce both transmission and translation latency.

Figure 2.25 shows the total transmission delay, considering various traffic load conditions. As it can be seen from the figure, the transmission delay is dependent on the traffic load condition at a given time. As the traffic load increase, the transmission delay increases. This is due to the added workload on the centralized translator implemented in our simulation. However, this may not significantly affect the translation delay, as the translation delay if an appropriate translator placement is performed. Dynamic translator placement problem could be considered as future work.

Since our proposed architecture provides the translator as a virtual function, there will be an additional delay due to the virtualization environment. Therefore, virtualization overhead delay has to be considered. Figure 2.26 shows the overall delay incorporating the virtualization overhead delay. As can be seen from the figure, depending on the traffic load, the end-to-end delay increases.

2.6 Emulation of LTE, LTE-A, and 5G Over a Lightweight Open-Platform: Re-configuration Delay Analysis

In this subsection, we provide a simplified functional deployment for LTE, LTE-A, and 5G function over a lightweight containerized environment. Network softwarization, containerization, and cloudification in a distributed and centralized environment are the current tread in 5G, Beyond 5G, and 6G. In that sense, significant activities are going on in the research community to softwarize the network functions deploying them in a cloud-native environment. Cloud-native architecture is an approach for network function and service to be built specifically to be deployed in the cloud. In this paper, we emulated LTE/LTE-A/5G in lightweight containers. We have evaluated the feasibility of using very lightweight environments such as k3s. We show the possibility of emulating a simple 4G/5G scenario without requiring a full Kubernetes infrastructure. The final results are based on open projects made accessible and used as inspiration as well as a starting point to then modify deployment techniques, configurations, and code. We have also explored scalability, orchestration/automation, reliability of the deployments. Finally, we measured and tested the performance. The performance evaluation shows the potential application of the open platform-based emulations and deployment in 5G and beyond.

The network keeps evolving adapting to the network user's dynamic demands. The constant evolution of wireless technologies brings new challenges. Moreover, several opportunities are also correlated to the continuous development and integration of new technologies. The network has gone through extensive changes to meet users' stringent demands. One main approach is network softwarization, which provides flexibility to enable the network to adapt dynamically.

In particular, the adoption of cloud computing and network softwarization is playing a very important role in network management and service provisioning. The two technologies are complementary to each other. In this regard, most applications and services are migrating from traditional hardware-based environments to remote and powerful cloud providers. Wireless networks are no exception to this migration. This has been facilitated by technologies such as software-defined networks(SDN), network functions virtualization(NFV), containerization, cloud-native, and microservices architectures. They are also expected to play a major role in the roll-out of 5G [14].

The arrival of 5G is increasing deriving the network complexity in terms of user density and stringent requirements such as ultra-low latency and reliability. To meet these stringent requirements, various approaches are introduced such as edge/fog computing. The introduction of edge and fog computing enables the possibility of deploying 5G network functions in a sparse and distributed environment, which are closer to the user to improve the network performance and user experience. However, this imposes further needs such as service orchestration and/or automation techniques to provide the required network services, creating a service function chain. It is expected cloud migration to be a top priority for the 5G core(5GC). Moreover, there is a growing interest in the wider availability of fully softwarized and open TestBeds environments.

We study the emulation of LTE and 5G over on open platforms. Specifically, this work focus on experimental evaluation by deploying and managing a completely virtual and open LTE and 5G TestBed. This helps to implement and exploiting different automation and orchestration techniques. In regards to the deployment, particular attention is given to achieve compatibility toward cloud environments.

2.6.1 Overview of Mobile and Cloud Technologies

This section presents an overview of different aspects of important background concept such as 5G, cloud technologies, and open software projects.

LTE Technologies

Long-term evolution (LTE) is the fourth generation mobile broadband network that attracted billions of mobile subscribers. There is still a demand that could even increase subscribers. LTE delivered huge improvement over the predecessor with the overall performance. It also paved the way towards more intensive softwarized network functions compared to previous generations. There are different versions of LTE. The most advanced and contemporary versions of LTE are known as LTE-Advanced and LTE-A pro. LTE incorporated inter-cell interference coordination(ICIC), multiple input multiple outputs (MIMO), the separation of control, and data plane known as control user plane separation(CUPS). This helps transition towards next-generation small cell technologies, such as network functional application platform interface(nFAPI). Other fundamental changes are eNB splitting (BaseBand Unit and Remote Radio Head), modification of core network (CN). The last releases of evolved packet core(EPC) have lost circuit-switched mode, allowing all components to communicate only through IP. This enables full packet switching. The main components of 4G EPC are described as follows:

- Mobility management entity(MME): is responsible for management and control of the EPC. It takes care of handover, paging, UE/eNB attachment. It also manages the necessary information to accept or reject access requests to the network.
- Home subscriber server (HSS): this function is responsible for the storage and access of users' information. It allows MME to have persistently archived all identifiers necessary to accomplish identification operations.
- Packet data network (PDN)-gateway (PGW): is fundamental for the correct functioning of EPC. PGW's main job is to allocate and assign IP addresses. It also allows access from EPC to the internet and the external world.
- Serving-gateway (SGW): S-GW is the complementary function of the user plane for P-GW, connecting radio access network (RAN) to EPC and granting access to PDN gateway.
- Policy and charging rules function (PCRF): it mainly manages different policies for user access, allowing differentiation between planes and QoS access.

The above functions follow the legacy deployment of 4G/LTE network, which is mostly hardware-based. This lucks the flexibility that the current network is requiring, especially in the era of network softwarization and cloudification. Moreover, an experimental tesbed is challenging to build since it requires using hardware-based functions.

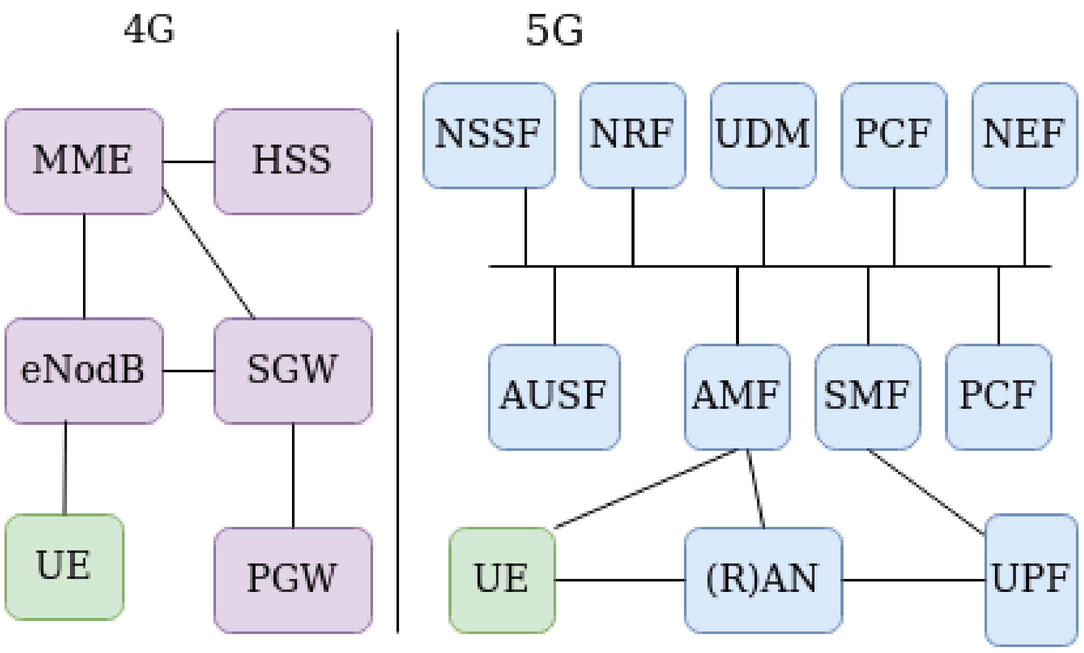


FIGURE 2.27: 4G/5G Core Network

5G Technologies

The first version of 5G commercialization has started to roll out with softwarization of functions. Its roll-out is facilitated by innovative possibilities derived from the softwarization and reduction of hardware. This enables the innovative solution to be developed to achieve the dynamic and heterogeneous demands of users. Since release 15 of 3GPP, which is the last version of LTE, all the developments are considered as an early version of 5G that links with LTE. Release 15 is more of a transition deployment, that allows new features but still being compatible with LTE-A. The major novelties involved in next-generation RAN (gNB) is the fundamental split between central and distributed unit (CU/DU). Moreover, there is also a separation between control and user plane (CUPS) with the dedicated entity for user plane function(UPF). Furthermore, there is an independent design and implementation of services such as RAN and CN. The 5G use service based architecture (SBA), utilizing micro-services for core component design. In general, 5G aimed at introducing the functions based on service-oriented approach, specifically, as microservice and multi-agent approach in service design[14].

Open projects on softwarized emulation of 4G and 5G are an emerging paradigm in the development of 5G. There are several alternatives. For the sake of research, it is preferable to consider open and free projects, which present anyway consistent and advanced opportunities. 5G components could be deployed in a virtual or docker virtualization environment. Examples of such works are OpenAirInterface(OAI) software alliance, Free5gc, Open5gs, SRS-LTE, and Labora projects are among several initiatives aimed at the progress in democratization and open deployment 5G deployment. These are an emulation of 4G/5G components in a containerized environment. The architecture of 4G/LTE and 5G are depicted in 2.27.

5G is expected to address the strange requirements of heterogeneous users. This can be addressed by creating more flexibility in the legacy network, paving the way for innovative solutions such as dynamic service orchestration and network automation[14]. In general, 5G needs to fulfill the requirements of the following categories of services: enhanced Mobile BroadBand (eMBB), consider high speed of end user data and system capacity as crucial, except for a slightly less focus on connection density and latency; ultra-reliable Low latency communications (URLLC), latency along with mobility will be crucial for reliability improvements; massive Machine-Type Communications(mMTC) main concern is the connection density parameter and network energy efficiency.

2.6.2 Cloud Based Wireless Technologies Deployment

An increasing number of network functions have been deployed in cloud environments. 5G roll-out is also expected to use cloud environments. Especially, 5G CN could be deployed as virtual network function (VNF) and/or cloud-native functions (CNF). Currently, there are several emulation environment developers to run complete architectures in cloud-native environments.

As indicated above, for studying and testing, there are several projects available with the aim of offering open and free access and actively implementing versions of their software that embrace cloud principles. This allowing developers to run on different local and clusters simulations of the core parts of LTE and 5G in the cloud. The previously described open-air interface software for RAN implements a version based on Cisco open/nFAPI standard, which is based on small cell forum protocol specifications. This is to achieve reduced cell dimensions for a granular coverage of particular cellular areas. This provides an advantage by the increase of the number of eNBs for the same area with a little management and configuration times.

There is a need for an easy and efficient way to reconfigure eNBs towards the correct version of CN in cloud management and orchestration of services. Deploying 4G/5G it is possible to measure and test the time required to reconfigure one or more eNBs while connecting them to a new core network. This requires relying on a cloud approach to the dynamic deployment problem. Measurement and analysis of the feasibility of the cloud environment and the subsequent deployment of the emulated 4G/5G component are necessary. Moreover, the performance evaluation of the deployments with specific KPI is also needed.

Kubernetes

is a tool for cloud and container technologies. It provides the mechanism for manage containers. It does not directly manage containers but it creates an entire structure to provide more advanced and specific abilities to the application requests. Kubernetes is an open-source project [60] that aims to automate the deployment and scaling of applications using an orchestrated management. The applications and elements in a cluster are the containerized, which has to run a container manager such as Docker.

Kubernetes has a central unit called a master node or control plane. It functions as an orchestrator and is used as the repository for the necessary information to maintain and restore application status in case of worker nodes failure. There are options with multiple master nodes to prevent the loss of the state in case of a master node failure. They are highly available clusters. The worker nodes are all the nodes other than the master node. The master node itself can be a worker node if it is allowed to run as a part of the application. Worker nodes have a couple of basic utilities to function correctly. They may vary accordingly depending on the features. The Kubernetes tool manages node and workload, which is also responsible for the communications between nodes and all utility operations. The second component is a container manager. The most common one is Docker which is delivered with several variants of Kubernetes. This component is responsible for managing and running all containers/pods that are designated for that locally worker node.

1. Pod: is the atomic component of Kubernetes. It composes the main functions of an application which can be scheduled, resources assigned. A pod can be composed of multiple different containers that can communicate with each other while working together to provide the pod function. Generally, pods or containers are managed manually once instantiated correctly;
2. ReplicaSet: pods are replicated to increase and scale a pod capacity. This is to ensure the right number of replicas are instantiated.
3. Deployment: through a manifest a deployment, its pods and replicas are defined. And then the deployment guarantees to restart failed pods and correctly manage replicaset to achieve the number of required instances.
4. Service: this is a separated entity and targets directly certain pods, allowing them to rely on or provide different "services". This is to connect them inside the cluster. It allows access from outside of the cluster networks.

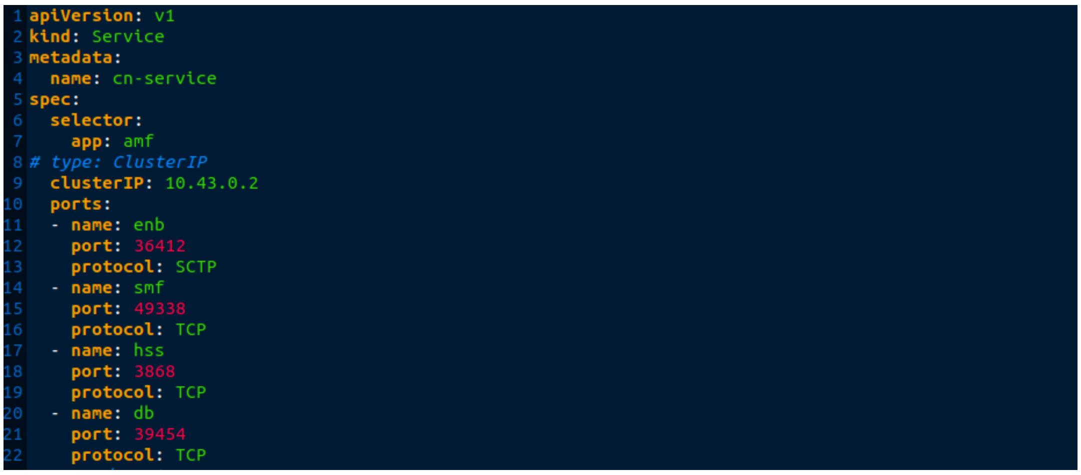
Docker can briefly be described as the direct manager of containers instantiated from deployments as pods, it is necessary on every node which requires to run containers. It can perform several different operations, like the management of container images and orchestration itself through other Docker tools like Docker Swarm.

Docker Hub is the set of sharing repositories for docker container images, it is also the source from which the Kubernetes cluster downloads images of containers once declared in the Yet Another Markup Language (YAML) manifests. Although there are alternatives to Docker as a container manager, probably the main advantage of using Docker is the massive amount of images present on Docker Hub.

The article's in [137] discussed the improvement and study of the reliability in the case of a virtual EPC(vEPC). The idea is to save the state of the running EPC with a hot backup to restore the core network to a configuration and status identical state before failure. The time required to restore a vEPC component is affected by the need for the eNB to be reconfigured to attach it to the new working vEPC. The service recovery time (SRT) in this case is 10s. Another crucial factor for the performance is the distance of the hot backup recovery vEPC. This showed the importance of having a solid and fast failure detection technique. Moreover, it also showed a responsive method to redirect served components to new and functioning replicas. This enables to save the state of the EPC and backup to the secondary deployment, by adjusting the target of the functional component serving the eNB.

Another work focusing on software deployment of 4G/5G architectures is presented in[13]. The paper focuses more on RAN aspects. It considers the switch between the two most advanced operating modes of eNB, the monolithic and the cooperation of a DU and a CU, known also as desegregated RAN. The paper showed another proof of the potential for a cloud-native deployment of 5G. In particular, it demonstrated the ability to switch between the two working modes using a custom openshift operator. This enables auto-reconfiguration of all the necessary component parameters to decide which RAN to use without changing the core network.

By far the closest work to ours is in [116]. It aims at deploying a fully automated 4G/5G environment, providing custom resource allocation based on specific parameters. Moreover, it accomplishes optimizing and predicting resource demands,



```yaml
apiVersion: v1
kind: Service
metadata:
  name: cn-service
spec:
  selector:
    app: amf
# type: ClusterIP
  clusterIP: 10.43.0.2
  ports:
  - name: enb
    port: 36412
    protocol: SCTP
  - name: smf
    port: 49338
    protocol: TCP
  - name: hss
    port: 3868
    protocol: TCP
  - name: db
    port: 39454
    protocol: TCP
```

FIGURE 2.28: Kubernetes service for both open5gs EPC and freee5gc 5GC

depending on traffic analysis. It is also possible to perform auto-scaling and de-scheduling procedures to orchestrate the number and typology of pods running in the cluster more efficiently. We focus on the deployment and usage of cloud degrees of freedom to implement and test novel approaches. The main objective of our work is to deploy and manage a cloud-native 4G and 5G system to study and perform possible exploitation of the cloud architecture.

2.6.3 4G/5G Emulation Testbed and Performance Evaluation

This work aims to measure the reconfiguration delay and resource usage for the 4G/5G emulation. Therefore, we compare the time needed to manually reconfigure an eNBs with a Kubernetes-based approach. The result shows a comparison for the selected Kubernetes cluster environment along with the implemented 4G/5G components. We also show the improvement provided in re-configuring the eNBs in a service-based cloud environment for different versions of core networks. The full deployment of 4G and 5G emulation is publicly available on Github that can be accessed using the links provided in the introduction section above.

Fast reconfiguration of 4G EPC and 5GC are necessary to capture the dynamic demands of users. The time required to reconfigure a high number of eNBs may significantly impact the performance of the network. To mitigate this problem, one approach could be to define a Kubernetes service for the deployment responsible for running the working core. And it could be possible to modify the target label of the core used, redirecting the eNBs traffic toward the new core version. However, we need to preserve the IP address to reach the core functionalities. The deployment of images from the Docker hub is easily supported if it is already functioning as standalone containers communicating through IP or port mapping. To have replicable and management configuration, we created YAML files manifest. It provides a pre-defined structure, defining all the necessary information to correctly start and manage an application inside the local Kubernetes cluster.

Creation of Kubernetes Cluster Using K3d

We first deploy a Kubernetes cluster. Then we deploy the 4G/5G emulations. We selected the deployment tools based on reliability and a certain degree of simplicity for

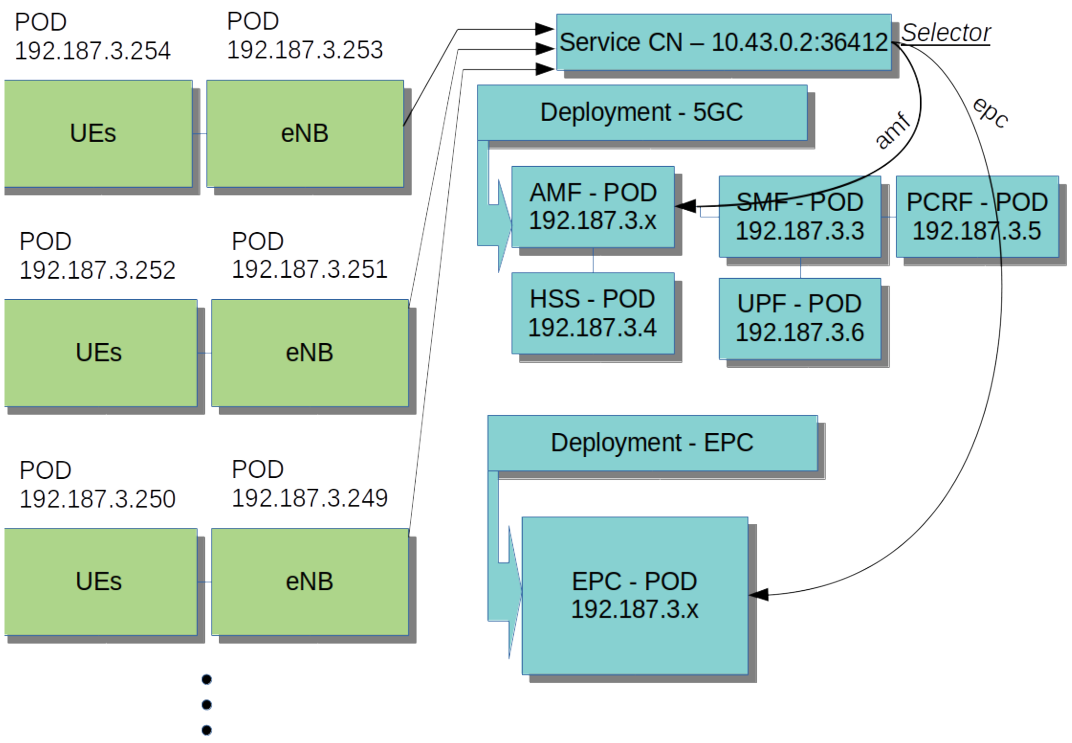


FIGURE 2.29: Final architecture for GEPC/5GC switch using Kubernetes service

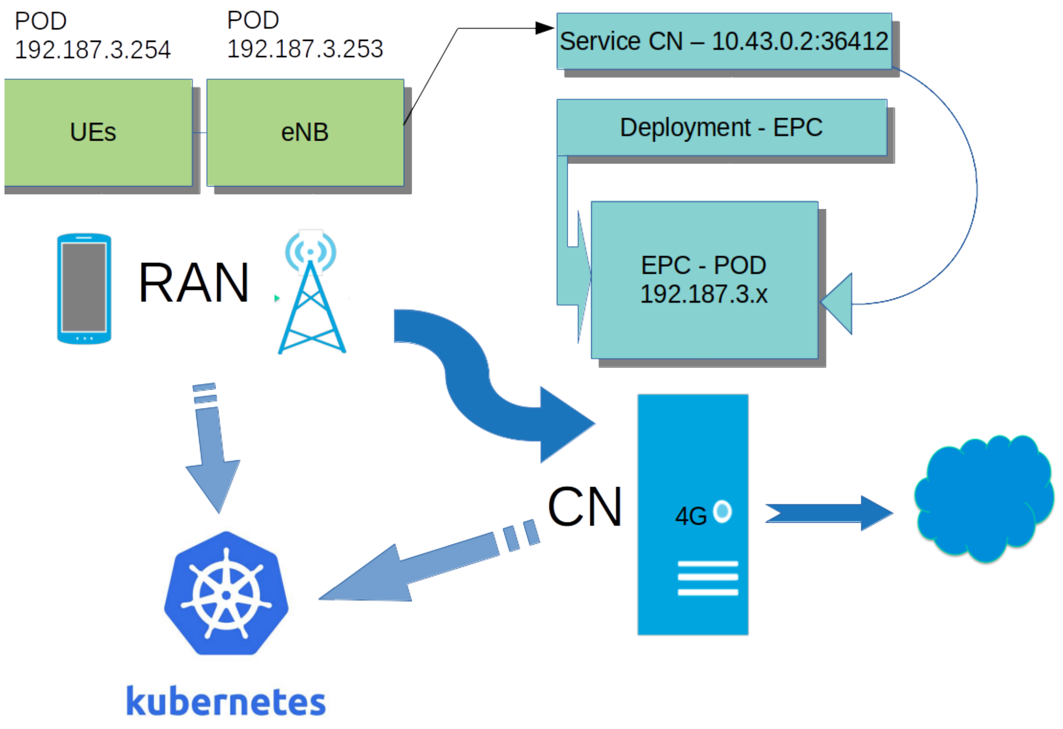


FIGURE 2.30: Components and their interconnection for 4G EPC

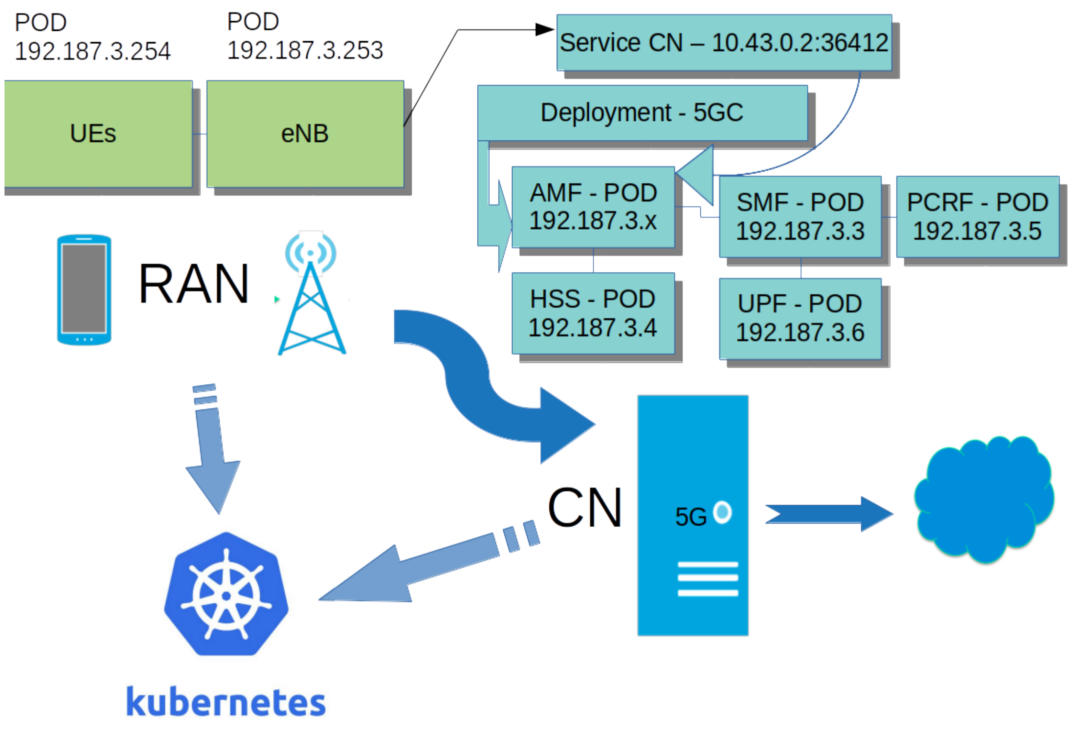


FIGURE 2.31: Components and their interconnection for 5GC

the management and configuration phases. First, we describe the installation, configuration, and management of a cloud-native environment for effective deployment of the migrated functions on a Kubernetes cluster. We used k3d v3.4.0, which is the first to allow stream control transmission protocol (SCTP) services which are essential for the exposure of some important ports of both EPC and access and mobility management function (AMF) services. The basic requirement for installing k3d is a correctly install, configure, and run the instance of docker. For the testbed created the release used has been docker version 20.10.0. Once docker is correctly installed and running, for which purpose the following documentation can be followed [78], it is possible to continue with the next steps; the recommended and supported version of the operating system is Ubuntu 18.04 LTS. Once everything is set up correctly we can proceed with the k3d latest version installation.

The installation of k3d can exploit several different commands and repositories, such as Homebrew and Arch User Repository (AUR) packages. To locally deploy a Kubernetes cluster, the first step is to emulate nodes as virtual machines or containers. The virtual machine nodes require high resources. They are also time to instantiate and run the full system. The two best containers are kind and k3d. Both are lightweight container nodes with less configuration burden in creating and configuring the Docker containers images. We selected the k3d and compared its performance with kind. We used the k3d v3.4.0. It allows SCTP services which is essential for the exposure of some important ports of EPC and AMF services. The basic requirement for k3d is to correctly install, configure, and running the instance of a Docker. We used the Docker version 20.10.0. Ubuntu 18.04 LTS is used.

The managed cluster is specified by the Kubernetes configuration file. It is placed in the home directory. IP addresses are assigned to specific components. This is necessary for the components such as the web user interface for the database and the mongo database itself. The pods receive a specific IP only when the calico networking solution [47] is configured for the cluster. Calico is an open-source project which is designed to provide network security for virtual machines and containers in cloud environments. Once calico is configured, it is possible to add some specific annotations to the pod manifests. This is to enable them to obtain an IP address from the Classless Inter-Domain Routing (CIDR) pool.

Migration of 4G/5G docker container images to Kubernetes cluster

We used LABORA project due to several advantages. It is easy and direct testing of all the functioning components. This is because it uses an Ansible file which is responsible for all required operations. The images are also easily accessible since they have been uploaded on the docker hub. Once properly tested all the components of the docker environment, we can migrate the eNB and UE. This is to start testing in the Kubernetes cluster functionalities. And then step by step all the manifests for each component have been written, allowing to completely migrate the testbeds to a cloud-native platform.

Once the Kubernetes cluster is created, the first step is to find and check a complete 4G/5G deployment. The used images are provided by LABORA project and the testing has been fast and optimized using a single ansible playbook for setting and launching the simulation. The first step was to test and manage the deployments in the docker environment, before moving towards the Kubernetes one. After obtaining the desired result, further customization has also be been performed. This shows the ability of the deployment to increase the number of UEs up to 10 instances. This is functioning with separated tunnel interfaces as well as the possibility to add multiple eNBs attaching and allowing UEs to connect through the shared core network.

The additional instances for both eNB and UEs, once the ansible file has been customized, presented the following parameters: the UE MSIN has been used from 0000000001 to 0000000010 and also the public land mobile network (PLMN) of a second eNB has been changed to 20894. To have the eNB recognized by the MME/AMF, it is necessary to modify the MME configuration file. It is also necessary to set a new entry with different PLMN and coherent Tracking Area Identity (TAI) and Globally Unique MME Identifier (GUMMEI). Moreover, the DB creation required modifications to include the new Mobile Subscriber Identification Number (MSIN) for the additional seven UEs. Finally, the UE configuration file is updated to recognize the new PLMN ID of the second eNB, together with modifying the eNB own configuration to correctly recognize the new identifiers. For correct functionality, the IP addresses have been assigned coherently. Once everything is tested for both attachment and connectivity, the process is included in the ansible file. This enables to replicate the single command for previous default emulations but running an increased number of UEs and multiple eNBs.

The first simplest test case considered for migration is the connection and communication of an eNB and a UE via an emulated channel and with a no S1 configuration. There is no core network attached. Thus internet access is not provided. To deploy the first scenario the eNB required the following manifest to be written and then deployed using the kubectl command. To check its functionality, a second pod containing the UEs emulation has been deployed and attached to the eNB performing ping executions to check also the correct functioning and connectivity of

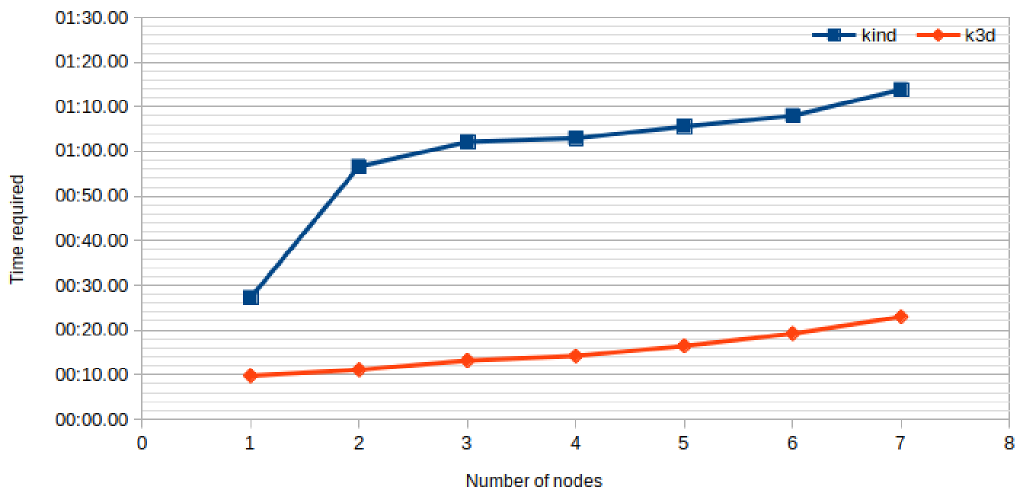


FIGURE 2.32: Comparison between kind and k3d

TABLE 2.3: Values for emulation components docker image size

CN component	Image size
UEs	7.04 GB
eNB	7.75 GB
EPC	883 MB
5GC	1.5 GB
MongoDB	500 MB
WebUI	450 MB

openairinterface software. A very similar manifest is written to deploy the UEs instances; changing name, IP address, and image. The images used are the LABORA images re-uploaded after the configuration to be able to directly run the simulations.

Figure 2.32 shows a comparison between kind and k3d based on the time required to create cluster. The advantage of k3d over kind is the stability and robustness.

As it seen from the Table 4.1 RAN consumes more resources. Figure 2.33 and 2.34 depicted CPU and memory usage of EPC. The resource usage change when the EPC components are instantiated and an eNB is attached to EPC through MME. Once the eNB has successfully attached 3 UEs is connected through eNB. Performing a ping operation to each UE slightly increase the resource usage.

Figure 2.35 shows the resource usage for eNB a single eNB, which requires more resources both in terms of CPU and memory. Figure 2.36 depicts the memory usage of eNB pod showing the variation in the simulation duration.

A final consideration is for the overall consumption of resources, considering the already specified different moments of the sample emulation. The overall impact of emulation are presented in the Figure 2.37 and 2.38.

As it can be seen from the Figure 2.37, the overall usage of CPU is coherent with the sum of single components usage. It is also possible to notice how the RAN emulation impacted the result. The memory usage also the increase is significant and coherent with the single components deployed and emulated.

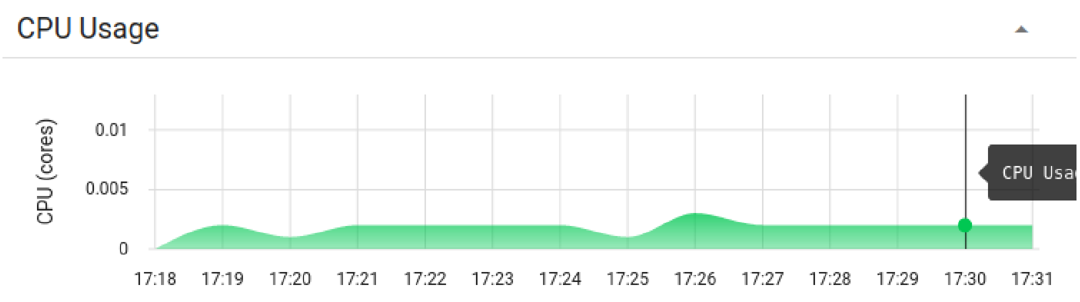


FIGURE 2.33: CPU usage of CN pod

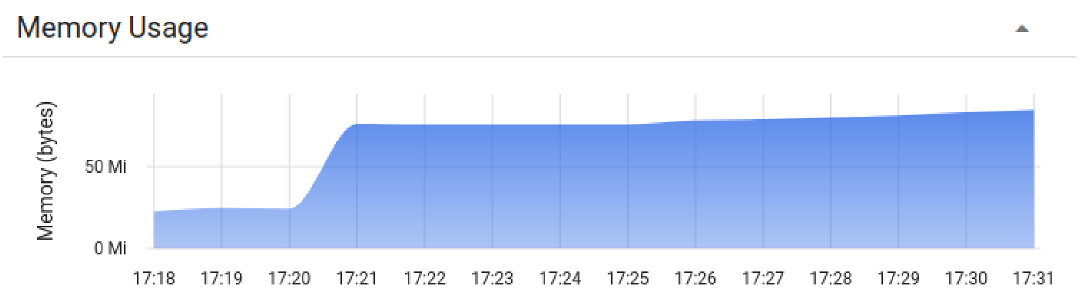


FIGURE 2.34: Memory usage of CN pod

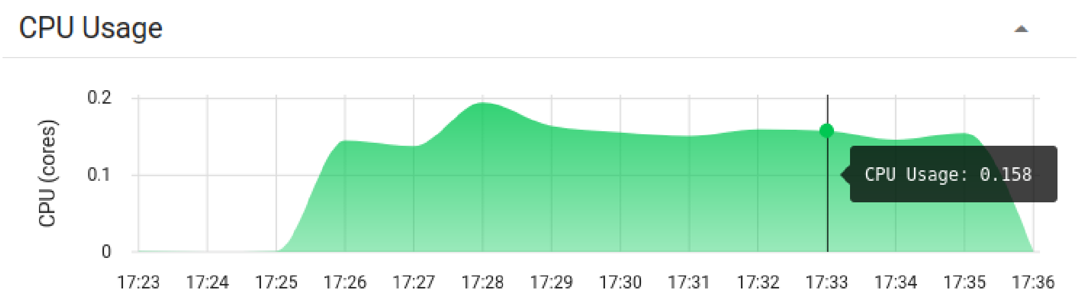


FIGURE 2.35: CPU usage of eNB pod

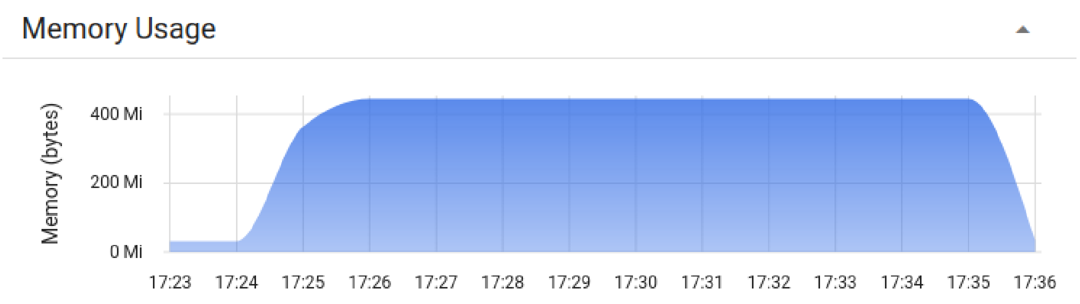


FIGURE 2.36: Memory usage of eNB pod

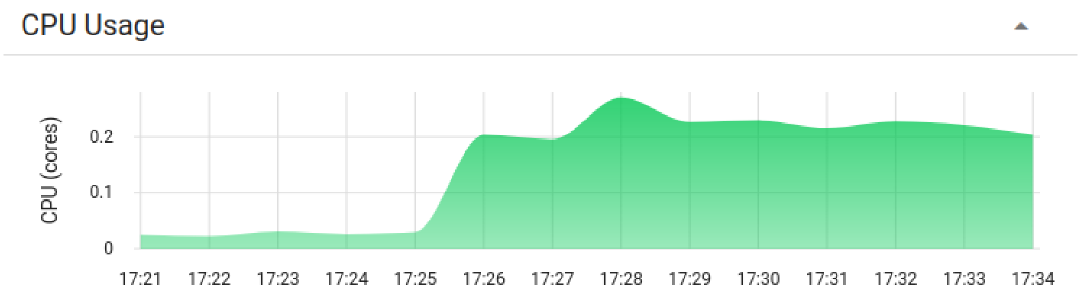


FIGURE 2.37: Overall CPU usage of kubernetes local cluster

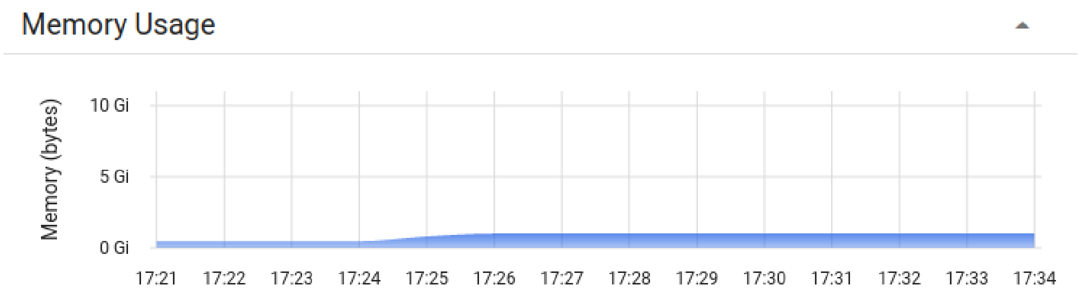


FIGURE 2.38: Overall memory usage of kubernetes local cluster

Chapter 3

Multi-Agent Based Network Management Automation

3.1 Introduction

Autonomic networking is the ability of a communication network to self-manage without (or with minimal) human intervention. In other words, it is the process of enabling the network system to have a self-X property; X can represent managing, configuring, healing, and protecting capability of the network. An autonomic management system has to be able to capture, respond and adapt to the dynamic and evolving behavior of the network, according to the users' and services' demands. In general, the primary aim of autonomic network management systems is not to completely exclude humans from the process. Rather it is aimed at shifting the burden of routine works, such as maintenance, configuration, and management, from network administrator to technology, alleviating tedious and repetitive tasks.

Traditionally, in the networking community, network management was typically performed either manually or by using pre-defined scripts to monitor and control software and hardware. Those solutions implement pre-defined strategies that are triggered by scenarios defined by the network managers. Recently, with the introduction of the concept of network softwarization, the capabilities of network management are enhanced by the programmability and adaptability of the networking infrastructure. However, while modern networks provide a high level of programmability, still automation in network management is supported by limited intelligence in the network.

Therefore, in the next generation networks, autonomic networking is required to scale up the network management capability to meet the expected big growth in the network introducing intelligence in the systems. This is because billions of devices will access and use the network. Thus, it will be extremely complex and cumbersome to continue managing future networks with the current practice. Moreover, from a business perspective, network operators aim at reducing capital expenditure (CapEx) and operating expense (OpEx), while increasing their revenues. Managing next-generation networks would require numerous highly-skilled manpower. Especially, with the arrival of the Internet of Things (IoT), and ultra-reliable machine-type communication (uMTC), there is a dramatic increase in the amount of network information to actively and proactively process. This requires a completely novel approach to network management. Therefore, 5G, and beyond networks [63] are trying to address this issue through some form of network management automation, alleviating the need for a huge number of skilled human network administrators.

Current networking researches mostly focus on network softwarization relying on SDN and NFV. These two emerging technologies enable the traditional static network to be flexible paving the way for network innovation. This opens the door for rapid network evolution and transition to intelligent networking, network automation, and smart networks. Network softwarization consists of mapping hardware-based network functions into software.

Moreover, SDN implies the physical separation of the network control plane from the forwarding data plane[97]. In the SDN architecture, the controller and the forwarding elements in the network are decoupled. This allows for a range of considerably more flexible and effective network management solutions using network programming. SDN enables the network to be programmed and dynamically re-configured, using software to outsource computing to external servers.

NFV is a software implementation of network function, which can run on general-purpose hardware[189]. The architecture of NFV contains three main components, such as

- Network functions virtualization infrastructure (NFVI) consists of the hardware and software that hosts different virtual network functions (VNFs).
- VNFs are the implementation of network functions such as firewall, network address translation (NAT), and packet/serving-gateway(P/S-G), and baseband unit (BBU) in software that could be deployed in an NFVI environment.
- NFV management and network orchestration (MANO) is the place where management and orchestration of VNF are implemented.

SDN and NFV are complementary technologies. Adopting them together provided myriad of innovative possibilities, such as dynamic network configurations, network states estimation and measurement and dynamic network management and control. They are enablers of network automation through the softwarization and orchestration of network functions. A unified architecture for SDN-NFV, which is presented in [148] is depicted in Figure 3.1.

Another enabling technology for network automation is machine learning (ML)/artificial intelligence (AI), which has recently seen greater advancements. This is expected to play a significant role in the automation of networks [43]. In fact, ML has been applied in various areas of networking such as traffic prediction, resource management, Quality-of-Service (QoS), and network security. ML provides cognition and reasoning capability in automated decision making. The goal of ML is to extract the hidden knowledge from the network, service, and users' behavior using historic data for its training and learning.

SDN and NFV enable the possibility of network programming, virtualization, and orchestration; however, they do not actually automate network management. Therefore, a new framework should be developed to incorporate SDN (for network control and programmability), NFV (for network function virtualization and orchestration), and ML (for knowledge management and cognitive ability) in a single architecture. Moreover, the current architectures typically use a 'monolithic process' in their implementations. However, in the monolithic process, there are strong dependencies between sub-functions. If one of the function fails, it affects or hinders the operation of other sub-functions[101, 279]. In addition, a monolithic SDN controller and an NFV Network Management and Orchestration Architecture (MANO)[189] result to be complex and difficult to scale. This is because a given sub-function can not be modified or scaled independently. Moreover, monolithic implementation of network management is not flexible, reusable, and scalable for complex systems. This makes it unsuitable for network management automation[101, 279]. Subsystem based approach has previously been introduced [101, 279]. However, their

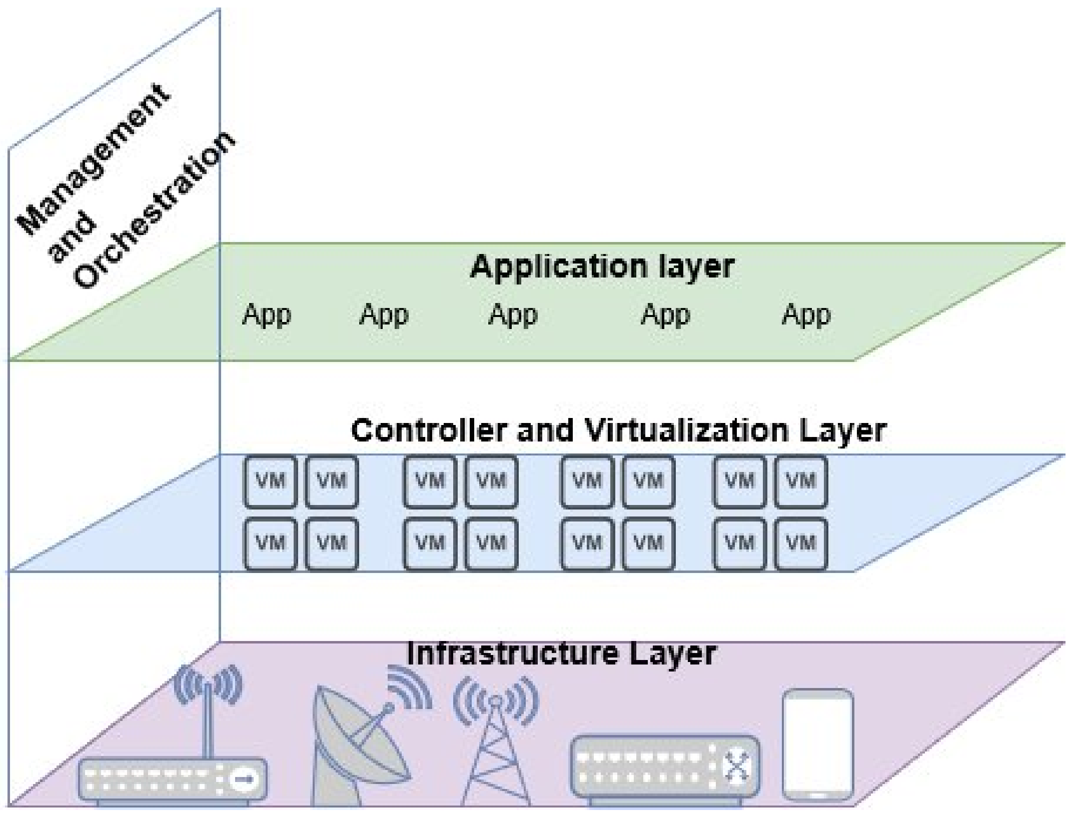

Management and Orchestration
Application layer
App
App
App
App
App
Controller and Virtualization Layer
VM
Infrastructure Layer

FIGURE 3.1: Unified SDN-NFV Architectural Framework [148]

discussion is limited to the sub-system creation and distributed deployment. They are limited in the introduction of intelligence in the sub-systems.

Given the above open challenges, this article proposes a MANA-NMS architecture. The core principle behind this architecture is network function division based on autonomic agents. The idea is to subdivide the monolithic network management and control function into sub-functions. It employs agents to be implemented in a virtualized or containerized environments[205, 285]. A given sub-function is deployed as an independent autonomous unit, called agent. Each agent is autonomous and specialized in performing specific network sub-functions, such as network slicing, path computation, and QoS monitoring. Agents are organized to perform the overall network management functionalities. Next, agents could be orchestrated with standard orchestration such as Kubernetes. However, orchestration could also be considered as a sub-function to be implemented as an atomic agent.

Specifically, the overall system behavior and functions should be represented, using multiple interacting autonomous agents. This requires a new architecture that defines the interaction, functionality, communication, and coordination between the agents in delivering the overall system functionality. In general, the overall system behavior needs to encompass different attributes such as agent communication, coordination strategies, decision making, learning, and cognition, in standard architecture.

We follow European Telecommunication Standards Institution (ETSI) standard's called Generic Autonomic Network Architecture (GANA) reference model[177]. The main goal of the GANA reference model is to prescribe the design and operational principles for Decision Elements (DEs) as the drivers for cognitive, self-managing, and self-adaptive network behaviors. ETSI is performing several GANA instantiations onto various targeted standardized reference network architectures, to enable autonomic algorithm implementation and to design and implement DEs in such architectures, in a standardized way. Our proposed autonomic architecture is presented in comparison with the ETSI's GANA reference model. The authors had already proposed a general architecture for autonomic network management (in terms of the autonomic operator) in [105, 106]. Moreover, they also described and extensively evaluated the self-management of a virtual network based on SDN, via the exploitation of adapted fuzzy cognitive maps at the SDN controller [20].

3.2 State of the Art in Multi-Agent Based Autonomic Networking

Automation is not a new subject since it was applied in several fields such as industrial automation, business process automation, and vehicular-driving automation. For clarity, we mainly divide the literature into two main categories: network automation and architecture, and the application of multi-agents in networking.

3.2.1 Autonomic Networking: Historical Perspective

Autonomics in communication networks can be reviewed considering state-of-the-art before and after softwarization. In fact, SDN and NVF along with cloud computing created a seismic change in the way we design, implement, operate and manage networks.

Autonomic Networking before the Introduction of SDN and NFV

Autonomic computing and networking were first envisioned by IBM in 2001 [133]. Inspired by autonomic computing, numerous research works had been done in autonomic networking since 2004 [260, 261, 259, 276, 29]. Some works also specifically focused on network management [276, 104]. In [276], the author proposed a system for autonomic network control and management. In [104], the authors presented an interesting approach to enable the self-X capability for network management. An early study of autonomic network architecture is discussed in [171, 42, 37]. In particular, Ghazi B. et.al. [42] aimed at designing a generic architecture by introducing the concept of abstraction. The main idea behind abstraction is to provide a platform for the coexistence of heterogeneous networking styles and protocols. In [37], the authors introduced an architecture called AutoXL. It adopts a knowledge plane (responsible for the management of knowledge) and a Policy plane, (responsible for managing policies). The study in [171] focuses on developing autonomic network monitoring for decision-making through information gathering, using a random number of monitoring devices. The work in [290] developed a specific network architecture for cognitive radio nodes.

There are also other specific network-based autonomic architectures and conceptual models. The works in [217, 24] mainly targeted transport networks. The authors in [217] presented a discussion on an attempt to enable self-optimizing and self-protecting optical burst-switched networks. Similarly, in [24] a model was developed using self-configuration, self-optimization, self-healing, and self-protection for service provisioning in automatically switched transport networks. The developed architecture aimed at enabling autonomic capability through self-governing systems.

Autonomic networking models for wireless networks are also a relatively well-studied subject[160, 186, 44, 8, 166, 82]. A specific work that focuses on wireless access networks, is presented in[238]. Specifically, it developed a distributed generic and autonomic management paradigm for decentralized management of wireless access networks. Considering the peculiar and inherent challenges of wireless networking, the authors in [160] provided an analysis of autonomic networking. In particular, they aimed at providing autonomic support in the wireless network design process.

In [44], the idea of *function atomization* was presented. Moreover, a consideration of composing autonomic elements was the scope in [8]. This composition aimed at providing a self-governing management system for overlay networks and a self-organizing composition towards autonomic overlay networks.

As per the authors' knowledge, the work in [166] showed the first use of agent-based modeling for automated elements. The authors argued the use of network management automation for wireless sensor networks as a self-recovery model. An interesting conceptual contribution could be extracted from the above works, which is *network element atomization*. This provides the possibility of sub-dividing monolithic network management systems into sub-units. The monolithic network management design has drawbacks such as reusability, scalability, tight coupling between components. Similarly, with a focus on a particular network type, few works attempted to incorporate autonomic in the network operation and management[82, 172, 170, 219].

Apart from an application on particular network scenarios, there are also works focusing on automating specific functionalities such as QoS monitoring, fault-management, traffic management, and security [306, 272, 191, 315, 182, 192, 33]. For instance, in [306], a discussion on a model-based approach is presented. The model-based method is to equip the fault-management system with autonomic capabilities. Article [272] studied automation of network operations for fault-management and resilience, while the article in [182] analyzed a memory-enabled autonomic resilient networking.

Autonomic Networking After the Widespread Adoption of SDN and NFV

SDN and NFV introduced a dramatic change in the legacy static network, paving the way for network software-based evaluation. From an architecture point of view, SDN provides centralized network provisioning and management capability, considering a global view of the network. Now, we focus on the most recent literature on autonomic networks that have been done since the widespread acceptance of SDN and NFV.

Ironically, after the introduction of SDN and NFV, there are relatively few organized efforts in the research community for network automation[257, 98, 317, 287, 177]. In [156], an experimental study on dynamic network reconfiguration in a virtualized network environment, using autonomic management, is discussed. An autonomic management system for a software-enabled network is also presented in [98]. The article in [159] presented a novel management architecture and elaborated the ongoing standardization efforts. Most of the concepts in the reference model, autonomic networking integrated model and approach (ANIMA) are heavily inspired by GANA architecture [177]. This architecture for IPv6 based radio access network (RAN) management. More recently, in[303], the authors presented a demonstration of an open-source MANO framework, an extension within the scope of the 5GTANGO H2020 project. They discussed automated service deployment, considering policy-based service level agreement (SLA) provisioning and monitoring. Aaron Richard et al. [225] discussed a data and knowledge-driven intelligent network for future network transformation. In [278], a first attempt to integrate SDN and autonomic network management was presented. This white paper shows the challenges and opportunities along with the autonomous network framework. An SDN-based autonomic architecture was envisioned in [218]. This architecture aims at enabling the coexistence of heterogeneous virtual networks, supporting hierarchical autonomic management. In [176], the authors introduced an architectural framework to be integrated with the network device operating system. It can work either independently on the device or can be aggregated across multiple devices in a network deployment.

Inspired by the cognition process of human beings, a cognitive network implements intelligent and adoptive procedures in responding to network dynamic behavior. The authors in [107] presented an evolutionary road-map of communication protocols towards the cognitive internet. They discussed the introduction of self-aware adaptive techniques combined with reasoning and learning mechanisms. The goal of these techniques is to tackle inefficiency and guarantee satisfactory performance in complex and dynamic scenarios. In [20], the author proposes a cognitive SDN architecture based on fuzzy cognitive maps. A specific design modifications of fuzzy cognitive maps are proposed to overcome some well-known issues of this learning paradigm. The authors also discussed the efficient integration with an SDN architecture.

Significant standardization efforts are also taking place by prominent standardization organizations such as ETSI and Internet Research Task Force (IRTF). The first project launched to develop a network automation model is Autonomic Network Architecture (ANA)[42]. The project aimed at designing an autonomic architecture to mitigate network complexity enabling protocols and algorithms to operate in an autonomic manner. Recently, ETSI has taken a prominent role in autonomic architecture standardization and adaptation. An Industry Specification Group (ISG) on Autonomic network engineering for the self-managing Future Internet (AFI) has been established under ETSI [302]. Various versions of this model were presented since its inception[177, 86, 87]. More recently, ETSI developed an autonomic network standard reference architecture[177], which is called GANA. In [51], SDN enablers in the ETSI AFI GANA Reference Model for Autonomic Management Control (emerging standard), and Virtualization impact.

In [50], an implementation guide for the ETSI AFI GANA model is presented considering a standardized reference model for autonomic networking, cognitive networking, and self-management. The document in [52] provides standardization of resilience survivability and autonomic fault-management, in future networks: this is an ongoing initiative recently launched in ETSI. As per the authors' knowledge, despite the tremendous effort to introduce an autonomic architecture and software model, none of the works have given enough emphasis on the concept of function *atomization*. Network function *atomization* and multi-agent-based representation of *atomized functions* are the core contribution of our work. Network function *atomization* is the sub-division of the monolithic process into smaller functions with cognitive and adaptive capability. With this added autonomic capability, sub-functions are represented as autonomic agents. Autonomic agents are *atomic units* that will be used as building blocks for autonomous network management systems.

Finally, the dawn of IoT and machine-type communications has dramatically increased the network size and complexity [281]. 5G and Beyond networks are evolving towards autonomic.

3.2.2 Multi-Agent Systems for Autonomic Networking

Now, we present a brief overview of the part of the literature emphasizing the application of multi-agent-based systems in networking. Multi-agent systems have been used in many areas [232, 128, 71]. A multi-agent-based configuration in a ubiquitous wireless network is presented in [130]. Where agents are used in the Radio Access Network (RAN). The information is loaded and fed back through agent communication. The Q–learning paradigm is used for the agent's cognitive ability to identifying whenever to make load balancing. In [71], the authors proposed the use of multi-agent systems within AuRA-NMS. They presented the use of multi-agent system (MAS) technology for distributed system automation and network control, considering the issues associated with robustness of distributed MAS platforms, the arbitration of different control functions, and the relationship between the ontological requirements.

Other works focused on more specific applications such as security, service discovery, and service migration. The article in [12] proposed an approach for dynamic service discovery in service-oriented architectures. The approach is based on a multi-agent system using the matchmaking technique. Patri et al.[211] developed a generic migration algorithm. This algorithm is derived from a search-based rational agent decision process. Such an algorithm can deal with uncertainties to provide the migration path. This migration is computed by using a maximized utility function. For military networks of computing and communication, the author in [139] presented an initial reference architecture. The reference architecture is a cyber defense application of intelligent software agents. Several works also exist showing the use of intelligent autonomic agents for security application such as risk assessment[180], and network intrusion detection [208]. The authors in [318] presented mobility management over the satellite networks based on virtual agent clustering.

None of the previews works considered an organized management architecture, like in [271, 155, 274], for the overall network system management. These works are closer to our approach towards utilizing agents, hierarchically organized them into specialized *autonomous-atomic units*. The work in [271] presented an attempt to organize a multi-agent hierarchical system for a self-adaptive network service. They discussed a hierarchical division of agents, effectively providing an abstract model, in which multi-agents are divided and organized into principal agents and sub-agents, according to different layers. Each principal agent is responsible for the management and coordination of sub-agents. Similarly, Guo-zhu et al. [271] presented a multi-agent-based management system designed for distributed network. They developed an intelligent management architectural framework. This architecture organizes agents into three types: center layer-agent, region-layer agent, and access-layer agent. An interesting work about control systems is presented in [262] from the control research community. Several existing works related to steady-states of the closed-loop systems studied state analysis of multi-agent-based system [242, 243]. Finally, an article developed a mathematical graph representation of agents' coordination [309].

Despite all the effort to develop a specific or general architecture and several applications of multi-agents, there is no work considering multi-agents for network management system automation, combining the concept of SDN, NFV, and ML in the context of the ETSI GANA reference model.

3.3 Network Management Automation Challenges

Future generation networks are very complex systems with numerous heterogeneous devices, users and services involved. However, they will be very dynamic and consistently advancing. Thus, managing such a dynamic environment involves handling heterogeneous real-time events in a sophisticated manner. This also means dealing with complex events, which require complex event processing. The problem is mainly complicated because of the huge amount of data generated by users, services, and network devices. Managing such an amount of data in such a complex environment, within a short period of time, requires a very tedious, enormous, and sophisticated processing. However, data also contains valuable information about users/services and network states even if it is cumbersome to manage them. Data could systematically be managed and analyzed to extract and use the information for self-management of the network. Extracted information could be used to facilitate services provisioning.

To automate a network management system, a comprehensive understanding of the complete cycle of the automation process is required. This includes measuring and understanding the environment, decision-making process, planning strategy, verification of the action plan, execution of actions, and/or finally monitoring the system behavior[105], as depicted in Figure 3.2.

- Measuring and understanding the environment determines a network behavior in a reactive or proactive manner, based on continuous observation of the constituent network elements. This enables the accumulation of network knowledge in the knowledge domain. For example, self-monitoring means having the ability to observe own internal state. Observation of this internal state can be a continuous measurement of QoS metrics such as latency, reliability, congestion, and throughput.

- A decision-making process is used to learn, understand and decide what action to take to respond to network events. For example, in a decision-making process, a machine learning model could be used to predict the incoming traffic, which is trained based on the previously accumulated data. Based on the prediction, a decision could be made on the type and quantity of resources required to satisfy the service's demand.

- Planning action strategy means executing the decision, such as what kind of actions or configurations should be performed on an element to have the required network behavior. For example, once the decision is made to allocate resources, a configuration file needs to be prepared to be executed in network elements or devices. This allocates the required network resources, which are decided in the decision-making process.

- Verification of action plan is the validation of the plan before executing it on the target network elements or devices. This could be a configuration file, which has to be verified and checked for accuracy and consistency.

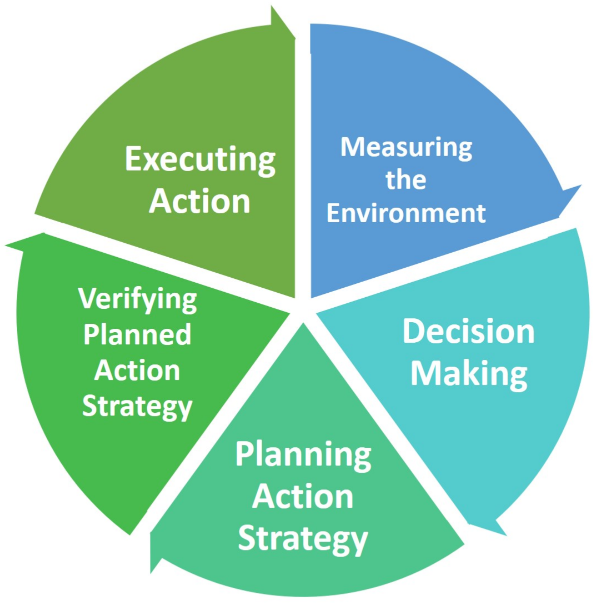


FIGURE 3.2: Network Management Cycle

- Executing action and/or output is implementing the verified or validated action plan. It could be a path configuration for the requested service or reservation of end-to-end network resources for service isolation, as part of QoS provisioning and guaranteeing.
- Monitoring the system's behavior completes the cycle. It continues to analyze the effect of the executed action on the overall system behavior while monitoring the environment for new events and changes. This could be through a loop-back control.

3.3.1 Decision Theory from Network Automation Perspectives

Decision-making means selecting one or more courses of action among the available alternatives. Humans make decisions every day. Humans have characteristics of weighing alternatives and deciding based on criteria such as good or bad (i.e quality). Decisions are generally characterized by gathering knowledge, learning, reasoning, and deciding distinctly. Gathering knowledge is the first step in the decision process. Based on the collected knowledge and experience, humans learn and reason. Humans also learn how to respond without an identifiable reasoning basis, just following instinct. For instance, a child may touch fire just to fulfill his/her curiosity. However, a reasonable decision is based on previous experience through pattern-matching. The human decision-making process is a long-studied subject in research and it is out of the scope of this article. Here, the aim is to get inspired by the human decision process to delegate intelligent decisions in network automation. In general, a decision is the key part of network management in the automation of processes. We aim to emulate and replace human involvement in the network management process[280].

In a similar analogy with the human decision process, intelligent-decision systems could have a knowledge accumulation step to observe and experience the network environment through parametric measurements. Besides, intelligent-decision systems could overcome human cognitive limitations and biases by providing a rational basis for comparison of alternatives[280]. The observation and experience are accumulated in a knowledge domain or *metaspace*. A cognitive process may modify the *metaspace* and the elements of its instances or models[280]. This could be through a learning process, by adding new rules, principles, and concepts to the *metaspace*, or manipulating and modifying the existing instances or models of the *metapace*. This goes with the assumption that the existing instances are already known. They could already be in the knowledge base or they could be added via new instances or models. The *metaspace* and instances constitute a knowledge base of the cognitive process.

A domain expert could be considered to provide knowledge to the knowledge acquisition module. That knowledge is encoded in the knowledge base, usually as part of the development process. The user or decision maker enters the system through an interface. Next, the user may directly access the knowledge such as a reference for past cases, or the inference engine to infer from past cases to a new case. The user may drill down for an explanation of the inference from the explanation module. The intelligent system could have a mechanism to capture, collect and infer knowledge from a domain expert, and pass that expertise to a decision-maker.

Generally, decision making is limited by available information and time to make the decision, together with cognitive ability and resources. Input to a decision-maker includes a *knowledge database* and *model base*. The database contains data relevant to the decision problem, while the knowledge base may include, for example, guidance in selecting alternatives. The model base holds formal models, algorithms, and methodologies for developing outcomes. Processing involves using decision models to simulate or examine different states and find the best solution under the constraints. Feedback from processing can provide additional inputs that may be updated in real-time to improve problem-solving. The output may generate forecasts and explanations to justify recommendations and offer advice. Results are provided to the decision-maker, who may interact with the system to provide additional information or query the results.

Decisions could be categorized as *structured*, *semi-structured* or *unstructured*. An example of a structured decision is the shortest path between two nodes that could be computed as a closed-form solution. However, if we are to provide a routing path for a packet or flow, then the shortest path may not be the best decision. It may not consider the current state of the network that could affect the routing such as congestion and available resources. The current state of the network could be so dynamic that we may not have the complete and exact network state information. This leads to a sub-optimal decision, which is considered semi-structured. In the unstructured decision, the problems have no agreed-upon criteria or solution and rely on the preferences of the decision making elements.

Depending on the time it takes to reach the final decision, decisions can also be categorized as instant decisions and long-term decisions. Examples of instant decisions are routing packets or flows from source to destination. Whereas, an example of long-term decision is the scheduling of system or re-configuring decisions for periodic updates.

In general, taking care of the network without administrators' conscious knowledge requires making an intelligent decision through decision generation, organization, verification, applying, and cheeking system behavior through loop-back control.

Data Analytics and Cognition

Managing the network is the ability of the network administrator or autonomic system to maintain the network within the desirable behavior. This is a very complex problem. It requires handling multiple and complex real-time events such as monitoring the network and computing resources which could be a physical channel (wired and wireless channels), encoding techniques, computing resources, stored network, and user data, users, service, and network devices (ports, links, configuration files) so on.

Reasoning and cognition are the key pillars in a decision-making process. Cognition is learning, analyzing, and reasoning to generate decisions. Useful information could be generated about users, services, and networks by analyzing the data. And then, this information could be used for reasoning out and generating management decisions, such as allocating resources, monitoring and guaranteeing QoS, authenticating users, and protecting the network from intruders.

There are various methods to extract information for given data. AI tools provide powerful aids in solving difficult applied problems that are often real-time, involving large amounts of distributed data, and benefiting from complex reasoning. AI tools could be fuzzy logic, case-based reasoning, evolutionary computing, artificial neural networks, and intelligent agents[280, 155].

Currently, machine learning is the most adopted means of extracting valuable information about users, services, and networks. With the interconnection of machine type devices, there is a colossal amount of data to learn from. The goal of machine learning is to extract the hidden knowledge about the network and users through training, using a network data set.

Generating Cumulative Decision

Communication networks will be very dynamic and complex. Handling events appropriately to maintain the desired behavior of the network requires making various intelligent and adaptive decisions. Autonomic generation of various decisions through data gathering, analysis, and intelligent reasoning are critical steps. These decisions are dependent on various internal and/or external factors. Events could be triggered by external factors such as service requests, device discovery, new service discovery, topology change discovery, traffic classification, traffic prediction (amount of traffic on each link may be for congestion control), traffic scheduling, fault or anomaly behavior detection, and security breach. Internal decisions could be the network update, scheduled re-configuration for network optimization, available resource monitoring, handling network device failure or misbehavior. All types involve complex correlations. For instance, network re-configuration for optimization of decisions may affect the QoS as it may involve a configuration that may limit the availability of allocated resources at a given time. The QoS for one service may affect the decision for another service, in case of limited network resources.

Therefore, in the network automation process, decisions have to be made considering other interrelated decisions to see the cumulative effect on the overall network behavior. Autonomous decision-making requires the generation of various decisions such as automatic event handling through scheduling of events and tasks. Moreover, the crucial part of autonomous decision-making is intelligent reasoning and cognition that as discussed previously above. That could be supported by AI/ML techniques.

Decision Organization

Networks systems consist of heterogeneous devices, which involve cross-layer complex-events such as signal transmission, processing, coding, decoding, routing, load balancing, slicing, QoS monitoring, resource management. Decisions at various layers have to be made and each decision has its peculiarities but also dependencies on one another. For example, the decision made to provide end-to-end QoS for a certain type of traffic involves decisions on protocols, the physical layer, the data-link layer, and the network layer. Moreover, a decision made at the physical layer could impact the once's at the data-link layer or network layer. Hence, abstraction is necessary to isolate and focus on particular aspects. Various approaches could be followed. One approach could be using a standard seven-layer OSI or four-layer TCP/IP model. Another one could be using a new customized version of these layers as followed in the ETSI GANA model. In our proposed architecture, we followed ETSI's GANA model for decision abstraction and organization[177, 155]

Overall Autonomic Decision Assessment and Verification

The outcome of an autonomic decision should be assessed according to both the process to and outcome of, decision making. The process of decision making is very critical. And it should be assessed since it has a big implication on the time it takes to decide and the additional computation and communication overhead. The quality of a final decision is also very important that should be assessed based on a defined criterion, which could be specific for a given management demand. Assessing a decision requires defining measurement matrices.

In the network management automation process, we need to have a mechanism to validate the final decision for consistency, accuracy, and optimality. The final decision's measurement and verification mechanisms should consider consistency in collective system behavior. That is because, in a cumulative decision, individual decisions may not be the best or optimal one, since multiple factors should be taken into considerations. It is important to model desired network behavior pattern for proper evaluation of the performance of the automation techniques. Typically, *healthy* network behavior initially is decided by a network service provider or network administrator, through a set of behavior constraints.

Verification mechanisms could be a mathematical procedure to verify and validate intelligent decisions for autonomic networking. It could be choosing the best solution from a set of, sometimes infinite, possible solutions. This may involve making sense out of ambiguity or contradiction in selecting the alternatives. A possible simple example of contradicting or conflicting decisions in autonomic network management could be a security alert with credible threats, such as a DDoS attack on the network that may require network reconfiguration for self-protection and attack mitigation. However, at the same time, the network may be serving critical users or verticals such as remote surgery or an automated car. An early act on the intruder could prevent or reduce the damage. However, the response may require some possible configuration scenarios, which could impact the critical services. Therefore, how to solve such conflicting situations has to be defined or acquired and developed through training, to be embedded in the network automation process. Finally, the generated re-configuration files have to be verified for the impact that it would have on the network (the accuracy of the intended outcome of the actions to be taken). In general, we need to verify and validate the final solution before applying it. And it is beyond the scope of this article to exhaustively discuss the validation and verification mechanisms to each type of decision.

Decision Execution

In network management automation; action or decision execution is needed to fulfill network management demands. An action could be re-configuration, database updating, path re-computing, authentication, authorization, and monitoring[105]. The outcome of decision generation needs to be applied to the system to change the part of the (overall) network system to fulfill service or management demands. The network's behavior change comes by applying the final verified decision to change the network environment so that it behaves as per the network management demands. The response to the change has to be done appropriately and timely. This is very critical. A simple example could be when we make a decision to route a flow based on the currently available path. The path status or available resource changes very quickly, which may invalidate or outdated the decision. This is mostly because, in network automation, the aim is to develop a context-aware response to the demands. Decision execution as apart of the automated process is also a vast and challenging area that needs to be explored.

Monitoring System Behavior

After implementing the final decisions, the network behavior has to be monitored to observe how the action affects the system behavior[105]. This is to evaluate the applied solution, which could be for consistency, correctness, and appropriateness. A typical mechanism that is applied in most automation comes from control theory, such as loop-back or feedback control. This also requires data gathering and data processing discipline, which correlates data from multiple sources to identify patterns of events that the network is expected to behave. Exploring this from an autonomic network management perspective is also a critical research area that needs to be investigated.

3.4 Multi-Agent Based Autonomic Network Management System

MANA-NMS architecture divides the monolithic management system into sub-functions to be performed by autonomic multi-agents. It can be considered as a way to organize parallel computing or task execution, as it includes the division of a complicated task into several more simple ones. The tasks are performed by the group of agents autonomously. In other words, we aim to provide a systems architecture called MANA-NMS which partitions a complex network management system into *autonomic units* or modules. These autonomic units are *atomic units* capable of performing tasks autonomously while interacting with the environment as well as with other agents. Modularity also helps in reducing network management complexity through the independent implementation of sub-units and components. These units are reusable.

As previously mentioned, we focus mostly on proposing general principles and directions to follow for automated network management. Moreover, the proposed

MANA-NMS architecture requires to deal with a vast area and numerous challenging topics. Therefore, studying all potential areas and topics including implementation technologies are beyond the scope of this paper. Nevertheless, our article attempts to investigate the feasibility of the proposed architecture through mathematical analysis of a preliminary representative system.

3.4.1 Multi-Agent based Network Management System

A network management system is a complex system having multiple events such as multiple-input, multiple-processing, and multiple-output or action. We identify that multi-agent systems are the best candidate in developing such complex, distributed, large-scale heterogeneous systems[128]. In addition to automation, the proposed MANA-NMS architecture provides modularity, re-usability, scalability, robustness, and re-configurable.

As a set of autonomous entities, agents are capable of performing a given task autonomously. Agents have the capability to replicate a single or multiple collaborative decision-makers. In a multi-agent system, agents are the smallest building blocks. The blocks are specialized in performing specific functions. Using the appropriate set of autonomic agents, it is possible to build a complete multi-agent system to replace the complex and distributed monolithic systems like network management systems. The constituting autonomic agents can communicate, collaborate, cooperate, and coordinate to accomplish complex tasks. In building a multi-agent system, several challenging aspects, such as autonomy, collaboration, activity, reactivity, communicativeness, and goal setting, have to be addressed.

Agents could be distributed to perform distributed tasks or assigned to handle different functions in a localized and specialized rule. Or they could be centrally located to perform the tasks arriving into the central computing or task performing multi-agents. Agents could also be homogeneous or heterogeneous. Homogeneous agents are agents with identical characteristics. They could be designed or instantiated to perform workload sharing. They could collaboratively perform large tasks that could be sub-divided into sub-tasks to be assigned to different agents. Moreover, agents could also be organized hierarchically [271, 155]. They could be organized to handle different functionalities.

An example of a multi-agent based system for network automation could be to perform network management functions. For example, some agents could be designed to monitor and analyze the network parameters that are required to provide given services or tasks. Other types of agents could be designed and dedicated for QoS monitoring for a given type of service. Some other types of agents could be decision aid or classifier or RAN slicer. Each of them is specialized in performing a specific task as per the demand. Finally, the overall system could represent an automated network management system.

A multi-agent system has been applied in implementing complex software systems[128]. They are capable of automating heterogeneous and complex real-time systems. A network management system is a heterogeneous complex system with real-time and complex environments. Therefore, we propose that a multi agent system is an effective approach for the network automation process. However, several challenges need to be addressed to exploit the solutions offered by this approach. This includes decision generation which involves environment monitoring, data analytic, cognition, complex event processing (CEP), and context-awareness.

Design Principles	Microservice	Multi-Agent System
Autonomy	Require no human intervention	Require no human intervention
Interaction	Interact with other microservices using RESTful APIs and HTTP	Interact with other agents using ACL
Re-activity Response	Respond to HTTP requests in a timely manner	Measure, understand and perceive the environment to respond in a timely manner
Pro-activity Response	Unable to take the initiative	Respond and take the initiative
Loose Coupling	Monolithic systems are decomposed into loosely coupled sets of highly cohesive co-located services	Autonomous and loosely-coupled solution providers

TABLE 3.1: Microservice and Multi-Agent Comparison

3.4.2 Multi-Agents and Microservices for Decentralized and Loosely-coupled Softwarized Systems

The industry is heading in the direction of highly decentralized and loosely-coupled softwarized systems such as distributed NFV and mobile edge computing (MEC). Currently, large-scale software development is based on microservices. A microservice is a model, in which systems are built from small and loosely-coupled services. They have the potential to be a tool for building systems such as network management systems. They adhere to the Isolated state, Distribution, Elasticity, Automated management, and Loose (IDEAL) coupling design principles. Multi-agent systems have many similarities with microservices [205, 285]. Their core similarity is related to independence and loose-coupling.

Despite their similarity, agents have an advantage over microservices for dynamic network management system's design. Agents have the autonomic capability to provide multi-agents the advantage in building automated systems. Recent work presented in Collier et. al. showed the possibility of combining microservices and multi-agent systems[285]. Table 4.1 presents a comparison between microservices and multi-agent systems.

3.5 Proposed MANA-NMS Architecture

Currently, there is no universally accepted definition for what is an agent. We define every component in the network management system as agents, in our context. We propose agents to be implemented with full autonomic capability, which is derived from agent-design principles. The overall network management becomes a multi-agent system. In principle, agents are assumed to have multiple algorithms such as cognitive and intelligent algorithms, which could be based on ML or Deep Learning (DL), computational intelligence, and other intelligent algorithms. This provides a cognitive ability for agents in their autonomic operations. In general, the internal components and specification of agents vary depending on the functionality and type of tasks or services they are designed to perform. A specification of a standard agent is discussed later in Section 3.6.

3.5.1 Network Functions *Atomization*

To subdivide and organize functions into agents, let us first elaborate the concept of network function *atomization*. *Network function atomization* (*atomic* agent in our context) is the idea of organizing autonomic functions at different abstraction level. This with the obvious reason of separately dealing with each layer. Functional unit representation is based on autonomic agents. The abstraction is based on the GANA model. Network management functions could be physical-layer, device-level, and network layer functionality. Physical-layer functionality (low-level *atomic* actions) could be network resource block (channel) allocation or physical transmission and reception of frames, error correction, data framing, etc. Device-level functionality could be packet/flow forwarding etc. Network-level functionality (high-level *atomic* actions) could be end-to-end QoS provisioning and monitoring etc. All those functions could be provided by autonomic agents with suitable internal components. Since agents represent various functionalities, they need to be modeled and categorized accordingly.

In a general category, the types of agents could be service-type or device-type agents. Alternatively, to align with ETSI GANA model definition of decision units[177], agents could be categorized into four types: protocol-level agents, representing protocol-level DE, function-level agents representing, function-level DE, device-level agents, representing device-level DE, and network-level agents, representing network-level DE. Each function in each layer could be represented as agents in the proposed architecture. All agents are specialized and dedicated to performing tasks autonomously while coordinating and communicating with other agents. However, we will limit the description to only network-level functions. The following are the most typical network-layer functions, that could be defined as agents:

- Topology Management Agents (TM-AG)
- Routing Management Agents (RM- AG)
- Network Slicing Agents (NS-AG)
- Service Function Chaining Agents(SFC-AG)
- Forwarding Management Agents (FM-AG)
- QoS Management Agents (QoS-AG)
- Mobility Management Agents (MM-AG)
- Security Management Agents (SM-AG)
- Fault Management Agents (FM-AG)
- Resilience and Survivability Agents (RS-AG)
- Service and Application Management Agents (SAM-AG)
- Monitoring Management Agents (MM-AG)
- Generalized Control Plane Management Agents (GCPM-AG)

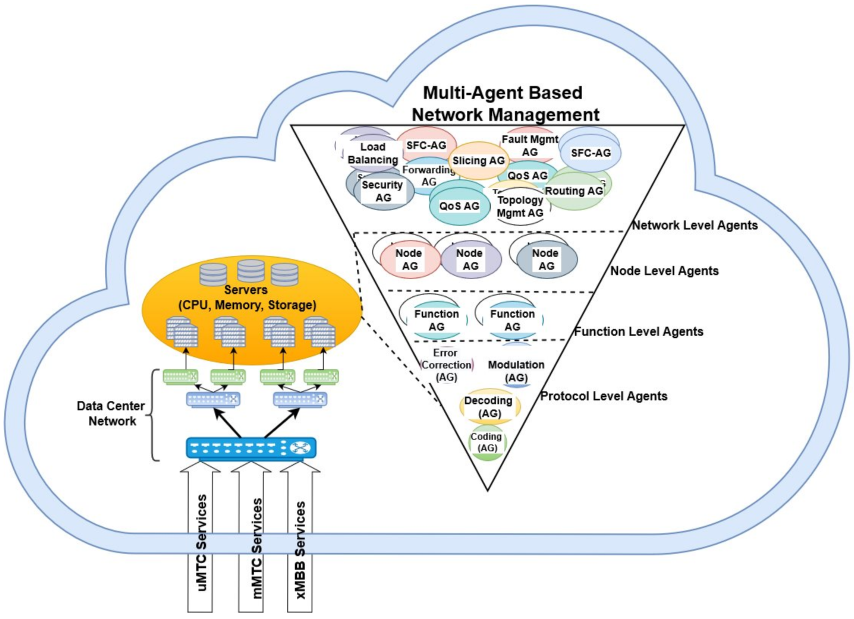


FIGURE 3.3: Multi-agent Based Representation of MANA-NMS for Cloud RAN

Now we focus on building the multi-agent-based autonomic network management system. Therefore, by using the above *atomic units* as a building block, we should be able to realize an autonomic network management system, replacing the existing monolithic management system. Figure 3.3 shows an example of a multi-agent autonomic network management system. The figure also shows functions' abstraction into four layers/levels/ protocol-level agents, function-level agents, node-level agents, and network-level agent.

In the process of building the autonomic management system, there could be a complex relationship between functions/sub-units that should clearly be defined.Each function performs specific service processing. In doing so, there could be an interaction of functions or units to accomplish the overall goal of service processing in the network management system. For example, when a given service arrives at the input of the system, it has a set of service constraints that have to be satisfied by the system for successful processing of the service. To perform that the functions or units collaborate or cooperate or compete to satisfy and perform the required service processing. This relationship could be complex in that in satisfying the service constraints, there could be contradictory requirements. This means that for instance to satisfy the reliability constraints of a given service while maintaining latency and resource constraints could be a complex optimization if we consider the overall system state. An example of the units composition relationship is presented in Figure 3.4. The overlapping of agents shows the close physical and/or logical proximity.

3.5.2 Proposed MANA-NMS Architecture in Comparison with GANA Architecture

As indicated above, agents are used as building blocks, replacing the network function to autonomously perform network management functions. Agents communicate by passing ACL messages to coordinate and they could share information with each other. Moreover, there are also different types of agents that should be organized in layers/levels. Figure 3.5 shows agents' layers organization in the proposed

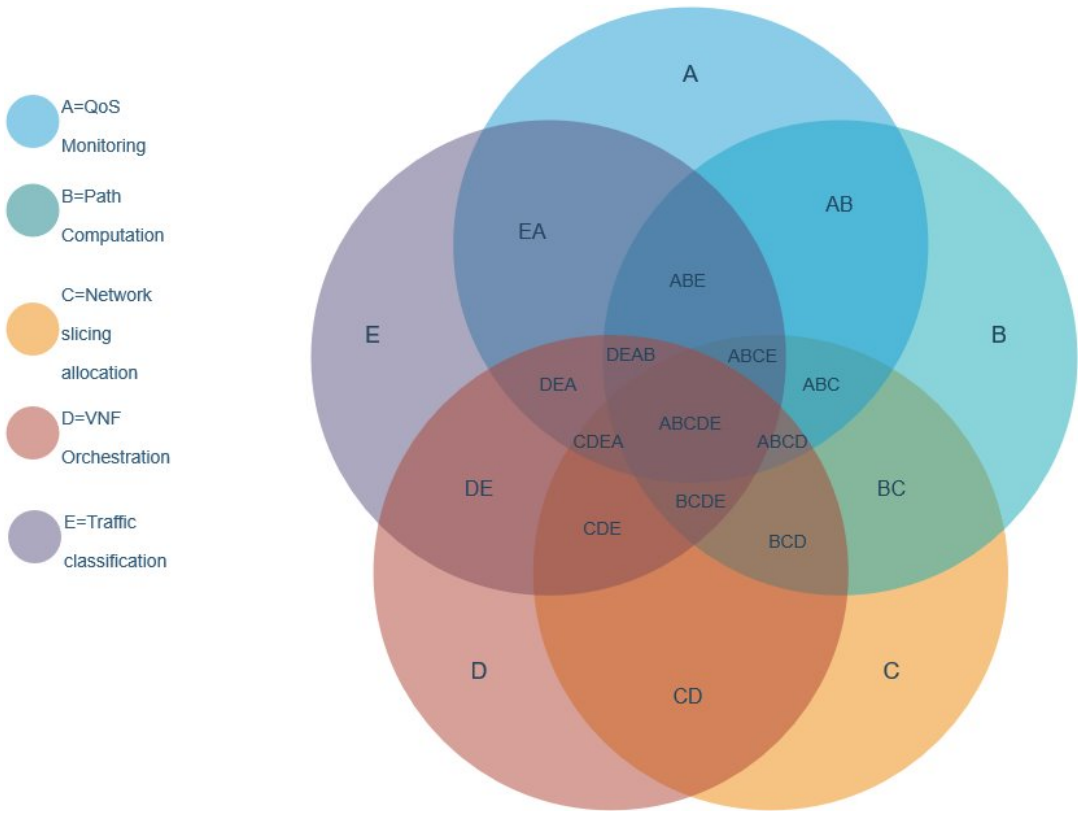


FIGURE 3.4: Simple Example for Network Level Agent Relationship

multi-agent-based architecture. The picture also depicts a comparison between the proposed MANA-NMS architecture and ETSI GANA architecture. Each agent represents an *atomized* network function such as QoS management function. Agents are expected to perform the intended original functionality of the network components or functional units. Multi-agents should represent all softwarized network components and functions as a system. Using the MANA-NMS architecture, agents are categorized as protocol level, function level, node/device level, and network-level agents as depicted in Figure 3.5.

Proposed MANA-NMS Architecture in Comparison with SDN-NFV Architecture

As discussed previously, SDN and NFV are the enabling technologies for network automation. The joint architecture is presented in Figure 3.1. Here, we present our proposed architecture in comparison with SDN-NFV architecture. Figure 3.6 shows a mapping of the proposed architecture in an SDN-NFV based architecture. As it can be seen from Figure 3.6, the management units are built using *autonomic units*. These autonomic *atomic units* are organized to cooperate and collaborate, and perform the overall network management functions. Moreover, as indicated in the above sections, the function of SDN and NFV could themselves be built using agents as the building blocks. Each component or modules of the SDN controller could be designed as *autonomic agents*. And the overall controller as a multi-agent system. However, a detailed discussion of this topic is beyond the scope of this article and left as future work.

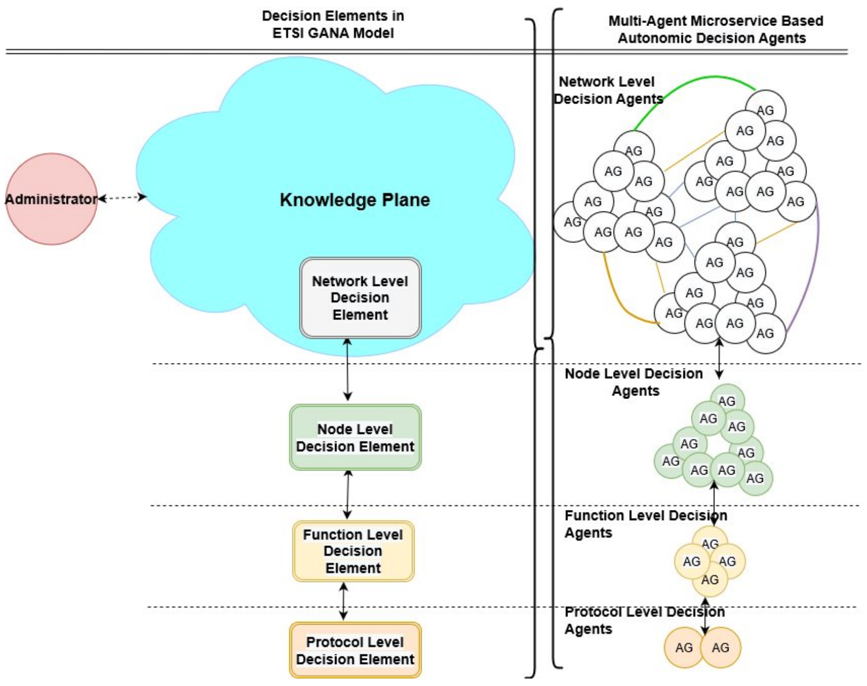

Decision Elements in ETSI GANA Model
Multi-Agent Microservice Based Autonomic Decision Agents
Administrator
Knowledge Plane
Network Level Decision Element
Node Level Decision Element
Function Level Decision Element
Protocol Level Decision Element
Network Level Decision Agents
Node Level Decision Agents
Function Level Decision Agents
Protocol Level Decision Agents
AG

FIGURE 3.5: Multi-Agent Based Representation of GANA Architecture.

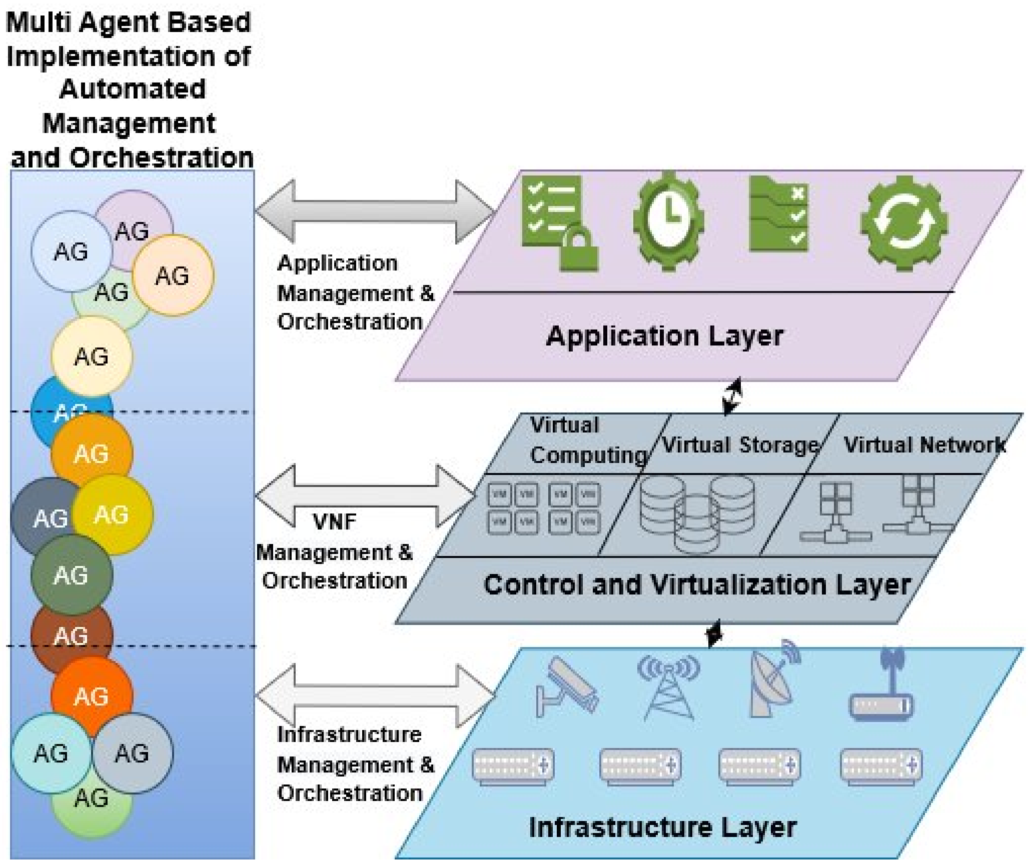

Multi Agent Based Implementation of Automated Management and Orchestration
AG
Application Management & Orchestration
Application Layer
VNF Management & Orchestration
Virtual Computing
Virtual Storage
Virtual Network
Control and Virtualization Layer
Infrastructure Management & Orchestration
Infrastructure Layer

FIGURE 3.6: Multi-Agent-Based Network Management Automation on Unified SDN-NFV Architectural Framework.

3.6 Mathematical Representation of Multi-agent Autonomic Network Management System

We proposed a multi-agent system for network automation. In the system, several agents are organized and coordinated to perform the network management autonomously, based on underlying control rules, laws, or policies. Developing multi-agent systems with a certain desired behavior requires analyzing the effect of each constituent agents to understand the collective system dynamics. There are few simulation tools available in the research, which could be used to analyze our MANA-NMS architecture. This is due to the complexity and dynamic nature of the network management environment. Therefore, we use mathematical modeling and analysis for the proposed multi-agent-based automated network management system to understand the collective multi-agent-based system behavior.

Let us start with characteristic description and specification of agents, which are considered as the *atomic units* of MANA-NMS. For the sake of simplicity and readability, here we characterize an agent with a very general specification. As depicted in Figure 3.7: an agent has an ID, for its identification; a *TYPE* describes agent category; an *INPUT*, is a service-request accepting and/or environment measuring points; *FACTS*, are knowledge database of the agent; *COGNITION (REASONING) UNIT* gives agents the reasoning capability; *PLANNING STRATEGY* organizes the steps or procedures for the action to be taken to satisfy the requested service; *VALIDATION* is the unit that verifies the action plan for consistency; and an *OUTPUT/ACTION* is the final decision (result or action) to be taken by the agent. An agent also has a mechanism to communicate with other agents through agent communication units. Following the definition of Genesereth, in [102], an agent is defined as a software component that is capable of exchanging knowledge and information. We consider the standard agent communication language called ACL messaging[253]. ACL provides a complex knowledge exchange mechanism such as facts, agent's goal, strategy, and plans.

First, we denote an agent Ag with a characteristic specification: $Ag = (A_{ID}, A_{type}, F, R, PS, V)$, where ID is the agent's identification number, A_{type} is its type which could be one of the following $A_{vh}|A_{prtcl}|A_{fun}|A_{nod}|A_{ntk}$. A_{vh} stands for virtualized or containerized agent type, which represents a hardware device such as switches or routers. A_{prtcl} stands for protocol-level agents. A_{fun} stands for function-level agents. A_{nod} stands for node-level agents. A_{ntk} stands for network-level agents. F stands for facts or lists of characteristics parameters that describe agents' knowledge base, which is extendable. The F list includes information such as the type of provided services and the desired delivery time. R is a reasoning mechanism such as machine learning techniques to generate decisions. PS is the action planning strategy such as configuration values and procedures to be performed. Finally, V is the validation technique along with the evaluation criteria to be used to verify the action plan for consistency and correctness.

We employ a stochastic approach for the analysis and mathematical formulation of the multi-agent system behaviour[149]. First, let us define a simplified agent states assuming agents with zero probability of failure. Le us assume an agent having four states: *start/input*, *internal execution*, *action* and *waiting* as depicted in Figure 3.8. This is a very general agent's state definition. In actual scenarios, each of these states may consist of a single state (action or behavior), or a set of states (behaviors). For instance, when an agent is in the internal execution state, it could be performing *reasoning*, *strategy planning*, *validation* or even *communicating* with other agent and

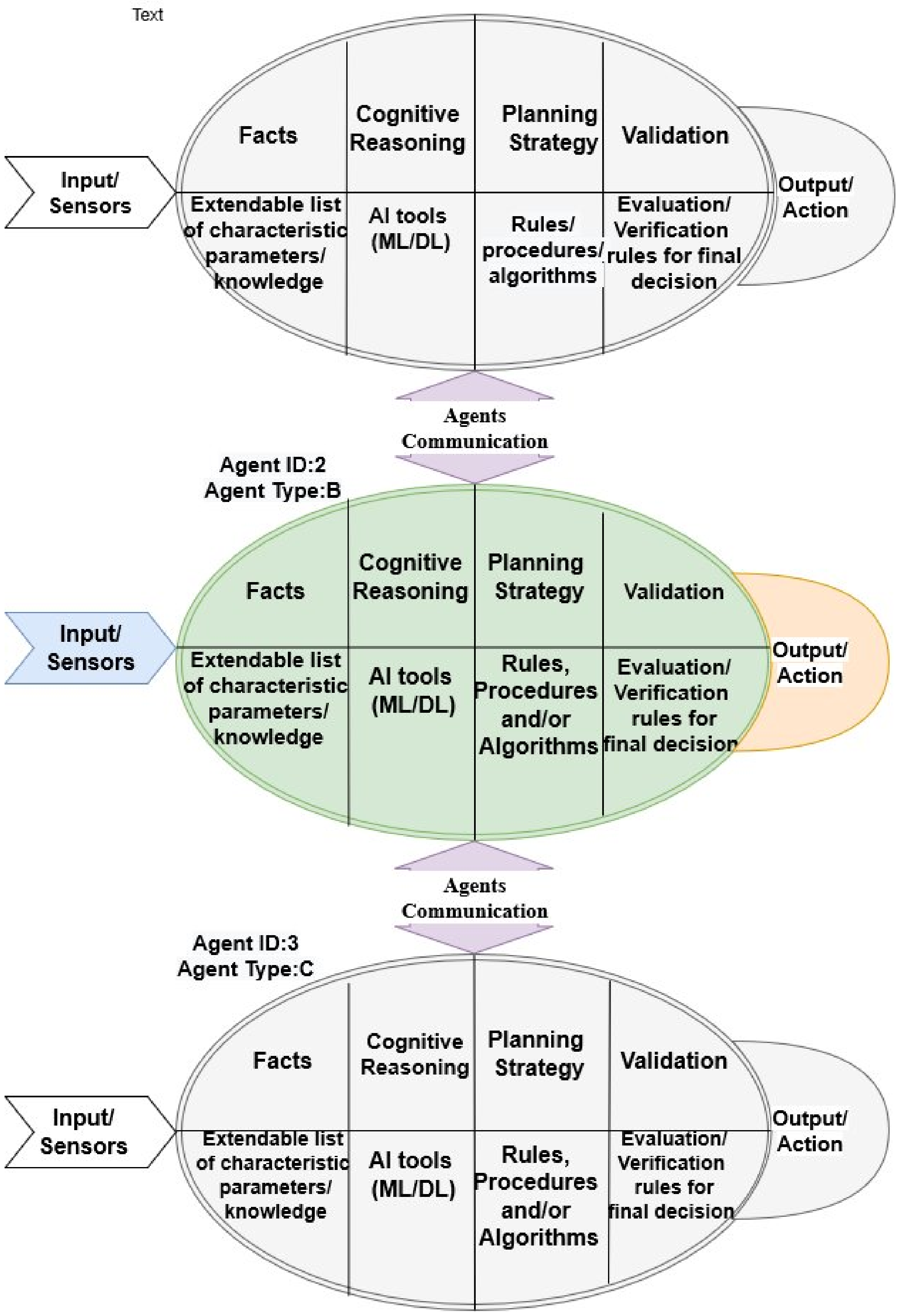

Text
Input/ Sensors
Facts
Cognitive Reasoning
Planning Strategy
Validation
Extendable list of characteristic parameters/ knowledge
AI tools (ML/DL)
Rules/ procedures/ algorithms
Evaluation/ Verification rules for final decision
Output/ Action
Agents Communication
Agent ID:2
Agent Type:B
Input/ Sensors
Facts
Cognitive Reasoning
Planning Strategy
Validation
Extendable list of characteristic parameters/ knowledge
AI tools (ML/DL)
Rules, Procedures and/or Algorithms
Evaluation/ Verification rules for final decision
Output/ Action
Agents Communication
Agent ID:3
Agent Type:C
Input/ Sensors
Facts
Cognitive Reasoning
Planning Strategy
Validation
Extendable list of characteristic parameters/ knowledge
AI tools (ML/DL)
Rules, Procedures and/or Algorithms
Evaluation/ Verification rules for final decision
Output/ Action

FIGURE 3.7: Fundamental Components of Autonomic Agent

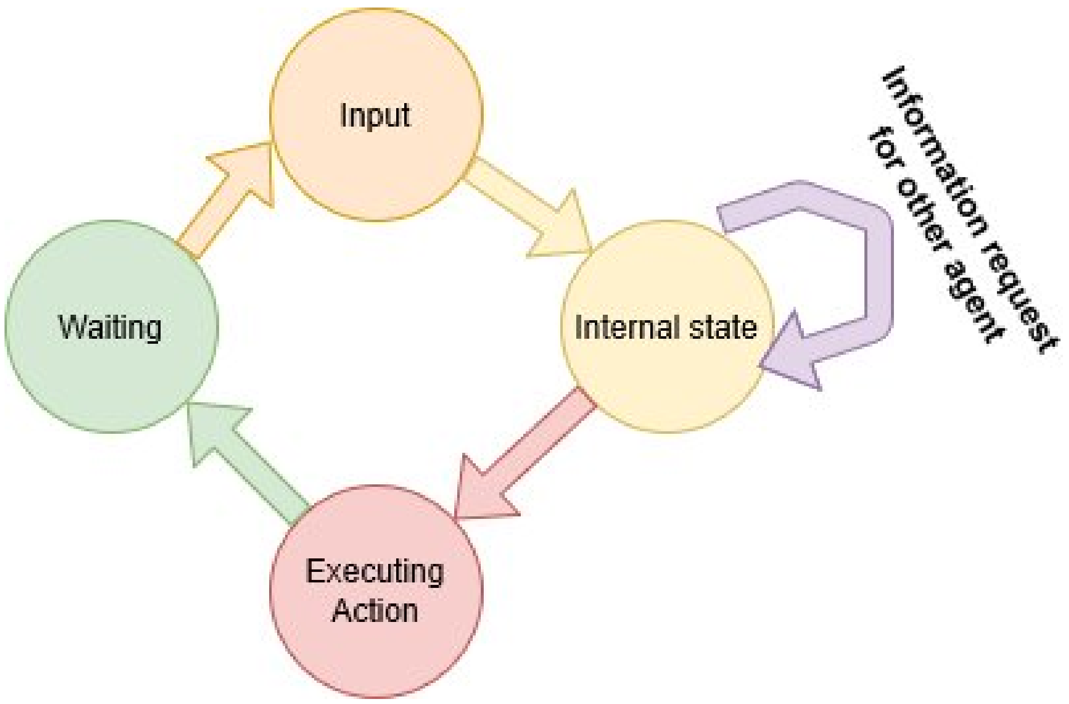


FIGURE 3.8: Agent States

waiting for responses. To find the collective system behavior of a multi-agent system such as system stability, steady-state, and network equilibrium, we assume the system as Markov process. Agents cannot meet Markov's property directly since they could use the memory of previous actions in decision making. It is commonly right to assume to include the contents of an agent's memory as part of its state to maintain the Markov property. Even if this may increase the number of states, many types of software agents and sensors do satisfy the Markov property with a reasonable number of states.

Before delving into the description, let us restate the Markov property; the configuration of a given system at time $(t + \Delta t)$ depends only on the configuration of the system at time t. Let $p(n, t)$ be the probability of an agent be in state n at time t. With this assumption, the change in probability density is written as [149]:

$$\Delta p(n, t) = p(n, t + \Delta t) - p(n, t) \tag{3.1}$$

The configuration of the system, which is defined by the the occupation vector is given by

$$N_k = \sum_{j=1}^{M} n_k \tilde{u} \tag{3.2}$$

where k is a discrete state with value $k = 1, 2, 3..M$. N_k is the number of agents in state k. The probability distribution $P(N_k, t)$ is the probability of the system being in the configuration N_k at time t. $\tilde{u}$ is the associated unit vector to each state. The transition rates for a Markov process, which is defined by the conditional probabilities, is given by [149]:

$$W(N_k|\tilde{n}, t) = \lim_{\Delta t \to 0} \frac{P(N_k, t + \Delta t|\tilde{n}, t)}{\Delta t} \tag{3.3}$$

The above equation describes the transitions to and from states in the configuration of the system, and it is known as the *Master Equation*.

Let w_ij be the transition rate from state j to state i then the *Rate Equation* is given by:

$$\frac{\partial n_k}{\partial t} = \sum_{j=1} w_{jk}(\langle N_k \rangle) n_j - \langle n_k \rangle \sum_{j=1} w_{kj}(\langle N_k \rangle) \tag{3.4}$$

The first term describes an increase in the occupation number n_k due to a transition to state k from other states, while the second term describes the loss due to the transitions from the state k to other states. This equation could be used as a tool to analyze the collective dynamics of an agent-based system. In simple terms, the rate equation is useful to study the system configuration and evolution in time.

A more useful system equation could be found if we consider agents as queue to model the overall system as M/M/1. Let us assume the service arriving at the system has a service arrival rate (λ_{tot}) and a mean serving rate (μ_{tot}). Therefore, in formulating the overall system dynamics, we use the overall system as an input-output system to analyze the resource and reliability constraint, serving capability, and processing latency.

3.6.1 Service Arrival, Scheduling, and Admission

Before presenting the formulation, let us define the incoming services to be scheduled for processing in the system. For the purpose of prioritizing service scheduling, we characteristically define services. Each characteristic and preference is represented D_i to be the *demand set*, which defines the requirements in terms of *latency*, *reliability*, *processing demand (workload)*, and *service class (priority indicator)* for that specific application. If the commodity is *elastic* $D_i = \{[\tilde{Th}_{max}], [\tilde{\tau}_{min}, \tilde{\tau}_{max}], [\tilde{\rho}_{min}, \tilde{\rho}_{max}], L_{ij}, \beta\}$. The first two requirements (latency and reliability) are ranges that represent services' *Type 2* and service *Type 3*, otherwise *inelastic* cases which represents service *Type 1*. The set becomes $D_i = \{\tilde{c}, \tilde{\tau}, \tilde{\rho}, \beta\}$, with constant first three members. The priority β is a value in the range $[0, 1]$, which is used to classify the serving priority of the commodity; the sum of all the priorities is 1.

Based on the priority, services are categorized into three type of services: service *Type 1 (inelastic)*, service *Type 2 (elastic)*, and service *Type 3* (with relaxed latency demand). In general, service traffic consists of all the three types of services. The total service workload composition and proportionality equation at a given time is given by

$$\begin{aligned} Srv_{tot} &= \sum_{i=1}^{N_{t1}} Srv_{t1} + \sum_{i=1}^{N_{t2}} Srv_{t2} + \sum_{i=1}^{N_{t3}} Srv_{t3} \\ &= \alpha_1 Srv_{tot} + \alpha_2 Srv_{tot} + \alpha_3 Srv_{tot} \end{aligned} \tag{3.5}$$

where $\alpha_1 Srv_{tot}$, $\alpha_2 Srv_{tot}$ and $\alpha_3 Srv_{tot}$ refers to Type 1, Type 2 and Type 3 services' workload in GOPS, respectively, N_{t1}, N_{t2} and N_{t3} refer to the number services of each type.

The overall service admission probability of C-RAN system is defined as the ratio between total admitted services and total requested services

$$Ap_{EDC} = \frac{\sum_{i=1}^{N_{srv-Adm}} Srv_i}{\sum_{i=1}^{N_{srv-adm}} Sr_i + \sum_{j=1}^{N_{srv-rej}} Srv_j} \tag{3.6}$$

where Ap_{EDC} is the total service admission probability of a given system (Edge Data Center in our case), $N_{srv-Adm}$ is the total admitted service workload, $N_{srv-rej}$ is the total rejected service workload.

Let us assume service arrival rate to be Poisson distribute. Using Poisson addition property, the total service arrival rate (λ_{tot}) at a given time is given by:

$$\lambda_{tot} = \sum_{i=1}^{Nt1} \lambda_i + \sum_{j=1}^{Nt2} \lambda_j + \sum_{k=1}^{Nt3} \lambda_k \tag{3.7}$$

where $N_{t(1-3)}$ service generating sources. We assume the critical determining factor for the system processing capacity is only dependent on the service processing agents, which are directly involved in service processing. The total serving rate of the system is given by

$$\mu_{tot} = \sum_{i=1}^{N} \sum_{j=1}^{M} \mu_{ij} = N_{Ag} U_{srv} \tag{3.8}$$

where N is the types of agents, M is the number of instantiated agents of a given type, N_{Ag} represents the total number of service processing agents that are required for a given service to complete its service cycle. We can calculate the mean number of services in the system using

$$\rho_{tot} = \frac{\lambda_{tot}}{\mu_{tot}} \tag{3.9}$$

The agents have finite resources to perform the required service processing. Hence, we need to consider this in the constraint equations. A constraint equation is developed considering the overall system to have multiple agents, working together performing the tasks. Following the M/M/1 queueing modeling, we can apply Little's Law, which states that the incoming rate of services has to be equal to the service rate to have a stable system with no rejected services. The overall load of the system must be equal to or less than the processing speed of the overall system. That is given by

$$\mu_{tot} \geq \lambda_{tot} \tag{3.10}$$

3.6.2 Service Latency in Edge Data Center

The overall delay in the system is the sum of mean delay in the serving agents and mean delay in the queues. In the model, the dependence of service processing and service transmission between cascaded agents is eliminated by approximating the arrival process of each flow at the subsequent agents as a Poisson process. This enables us to employ an M/M/1 queueing model to calculate the average processing latency. Therefore, the total delay is given by

$$D_{tot} = \sum_{i=1}^{K} Qu_D + \sum_{i=1}^{K} Ag_D \tag{3.11}$$

where Qu_D and Ag_D are mean delay in the agent and mean delay in the service scheduling queue, respectively. The above equation assumes all the required ordering sequences of agents. However, the real application varies from service to service, because of the possibility of splitting services between multiple agents for a faster process of a long single process.

Moreover, to calculate the total delay in passing through the required sequence of service agents, we sum all the delays incurred by the given service in the service function chain (SFC), assuming a queue at each stage of the Agent sequence. This is given by

$$\tilde{\tau}_{queue} = \sum_{i=1}^{N} \frac{1}{M_{sfc}U_{Ag} - \lambda} \tag{3.12}$$

where M_{sfc} is the processing capacity of an agent sequence (the number of agent types that a service require (N) multiplied by each Agents' capacity U_{Ag}), ignoring agents' CPU (overhead) requirements. That means all the allocated capacity of service processing agents in the data center would be used for service processing (without any other processing overhead), given by

$$M_{sfc} = \frac{CPU_{tot-srv}}{N_{Ag}} = U_{Ag} \times N_{Ag} \tag{3.13}$$

Therefore, to meet the end-to-end delay for a service that is constrained to pass through a given sequence of agents, the data center delay is the sum of SFC decision delay (overhead), queuing delay, processing delay at each agent, and interconnecting links delay (virtual links) between hosts, so that

$$\tilde{\tau}_{Edc} = \sum_{i=1}^{K_{link}} \delta_i + \sum_{i=1}^{N_{Ag}} \frac{L_{ij}}{U_{Ag}} + \sum_{i=1}^{N_{Ag}} \frac{1}{M_{sfc}U_{Ag} - \lambda} \tag{3.14}$$

where K_{link} is the number of links interconnecting different sequence of agents in the SFC. The above equation shows the total delay at edge data center.

3.6.3 Data Center Based Multi-Agent System Reliability

Finally, we provide reliability formulation for server and agents. The overall reliability is calculated by considering the reliability of server and agents to be mutually independent events[109]. A given physical server's reliability is given by [18, 39]

$$\tilde{\rho}_{phs} = \frac{MTBF_{phs}}{MTBF_{phs} + MTTR_{phs}} \tag{3.15}$$

where $MTBF_{phy}$ is mean time between failure, and $MTTRphy$ is mean time to repair. Similarly, a given Agent's reliability is given by [18, 39]

$$\tilde{\rho}_{Ag} = \frac{MTBF_{Ag}}{MTBF_{Ag} + MTTR_{Ag}} \tag{3.16}$$

where $MTBF_{Ag}$ is mean time between failure and $MTTR_{Ag}$ is mean time to repair. Therefore, the overall reliability of edge data center (EDC) is given by

$$\tilde{\rho}_{Edc} = [(\Pi_{i=1}^{N}\tilde{\rho}_{vlink})(\Pi_{i=1}^{K}\tilde{\rho}_{phs})(\Pi_{i=1}^{N}\tilde{\rho}_{Ag})] \tag{3.17}$$

To improve agents reliability, we may need to apply backup and overprovisioning of agents. If we apply backup for agents, the reliability increases. Therefore, the reliability of SFC incorporating backup Agents is defined as [220], which is given by

$$\tilde{\rho}_{Ag-with-backup} = \Pi_{i=1}^{N}(1 - \Pi_{i=1}^{N}[1 - \tilde{\rho}_{Ag})] \tag{3.18}$$

The reliability of edge data center considering the data center links becomes

$$\tilde{\rho}_{Edc} = [(\Pi_{i=1}^{N} Pr_{link})(\Pi_{i=1}^{K} Pr_{phs})(\Pi_{i=1}^{N} \tilde{\rho}_{Ag-with-backup}] \tag{3.19}$$

Using these techniques, maximizing reliability increases the total number of idle agents. Idle agents could be backup agents, which are waiting to replace working agents in case of failure or waiting to be scheduled in case of sudden and unexpected workload spikes. However, this requires additional physical servers to be activated. This increases the total cost due to power consumption and computing resource utilization. Therefore, we propose to use backup agents for those services that requires ultra reliability. In other words, we only use this technique for Type 1 services. We reserve some number of agents considering the required SFC for Type 1 services.

3.7 MANA-NMS Architecture's Performance Evaluation

This section provides a proof-of-concept for the performance evaluation of the proposed multi-agent autonomic network management system, using Matlab simulation. We rely on the mathematical formulation presented in Section 3.6.

3.7.1 System Description and Simulation Environment Specification

The simulation environment is aimed at emulating a MANA-NMS hosted in a cloud-based edge data center that autonomically manages a backhaul network and 5G service processing. A depiction of the simulation environment is presented in Figure 3.3. As it can be seen from the figure, there are three types of 5G services arriving at the edge data center. These services are ultra-reliable low-latency services (uMTC), machine-type communication (mMTC), and enhanced mobile broadband service (xMBB). The figure also shows an edge data center network that interconnects servers, which contain memory, CPU, and storage. In the edge data center, multiple types of agents could be instantiated to provide different functions. These functions could be service-traffic workload prediction, network-state database management, and synchronization, service authentication and admission, resource slicing and service scheduling, path computation and routing, service QoS monitoring, and security/firewall.

The evaluation is performed for a simple case of multiple types of agents, performing service processing and network management. For the evaluation purpose, we limit our consideration to only the following type of agents. All of them are network-level agents A_{ntk}.

- Service workload arrival prediction agents
- Event handler and service scheduling agents
- Service-processing agents, performing service processing such as authentication and admission, routing, QoS monitoring, and path computation for the arriving services
- Network state database updates, synchronization and management agents
- Service function chaining and orchestration agents

We categorized agents into two main types with the specification shown in Table 3.2. These two types of agents are system background service agents and user-traffic processing agents. Background service agents provide necessary preliminary

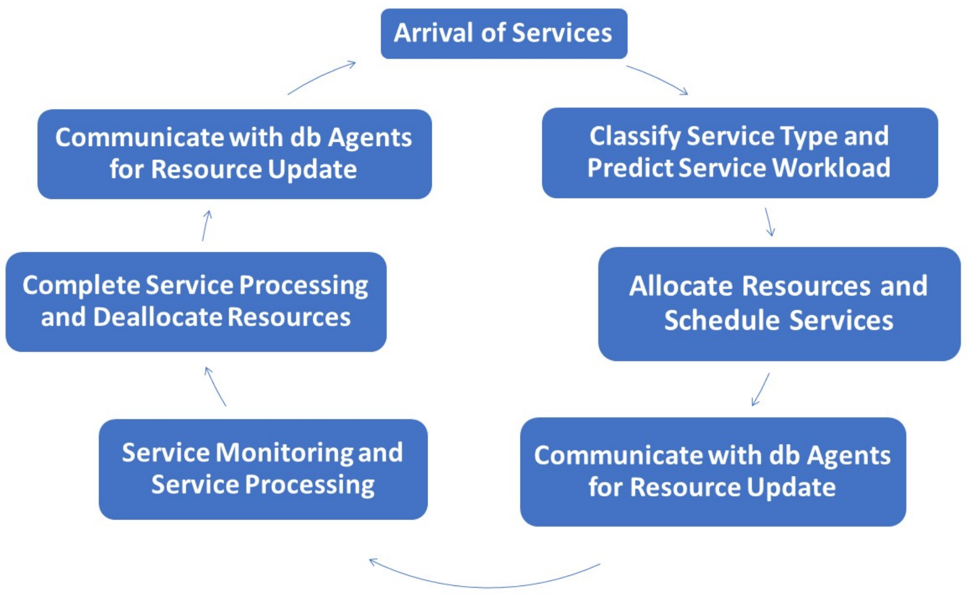

Arrival of Services
Classify Service Type and Predict Service Workload
Allocate Resources and Schedule Services
Communicate with db Agents for Resource Update
Service Monitoring and Service Processing
Complete Service Processing and Deallocate Resources
Communicate with db Agents for Resource Update

FIGURE 3.9: Cyclic Step of Service Scheduling, Processing and Monitoring

functions such as service traffic workload prediction, agents instantiation, and resource allocation, network monitoring, and service-agents orchestration. Such types of agents are assumed to have no direct impact on the service processing (eg. delay). User/traffic/service workload processing agents perform functions that are directly related to handling the incoming services. These could be path computing, authenticating, admitting, and schedule while monitoring a given service's QoS requirements for processing in the edge data center. Figure 3.9 shows a complete cyclic step of the overall multi-agent-based autonomic service management and processing system. The figure shows the steps needed to accomplish the scheduling, processing, and monitoring tasks for a given service. Note that the simulation system is built based on the proposed principles that the atomic agents are used as building blocks for the automated network management system.

As discussed above, one of the agents is an event handler that responds to incoming services or any network-related events. Based on the events, it reacts and dispatches the events (i.e arrival of tasks, services, packets, network changes, etc) to appropriate agents for event handling and processing. There are also traffic prediction agents, which predict the incoming service workload. Based on the predicted amount of workload, the required number of service processing agents are instantiated, and service function chains (a sequence of service processing agents) are created. In the simulation, this is performed proactively, every hour. Based on the predicted workload, additional agents are instantiated if an increase in workload is predicted in the next hour compared to the current period. Or some agents are destroyed to reduce and free edge data center resources if a decrease in incoming service workload is predicted.

Depending on the service requirements, the service agents are sequenced as a service function chain for a given service to pass through as a part of service processing. We considered four service agents as a control plane and data plane functions[15]. Since the data plane service function chain normally would not span more than a few service agents, the evaluation considers chains that are limited in size (four agents). An example of agent sequencing is depicted in Figure 3.10. The service function

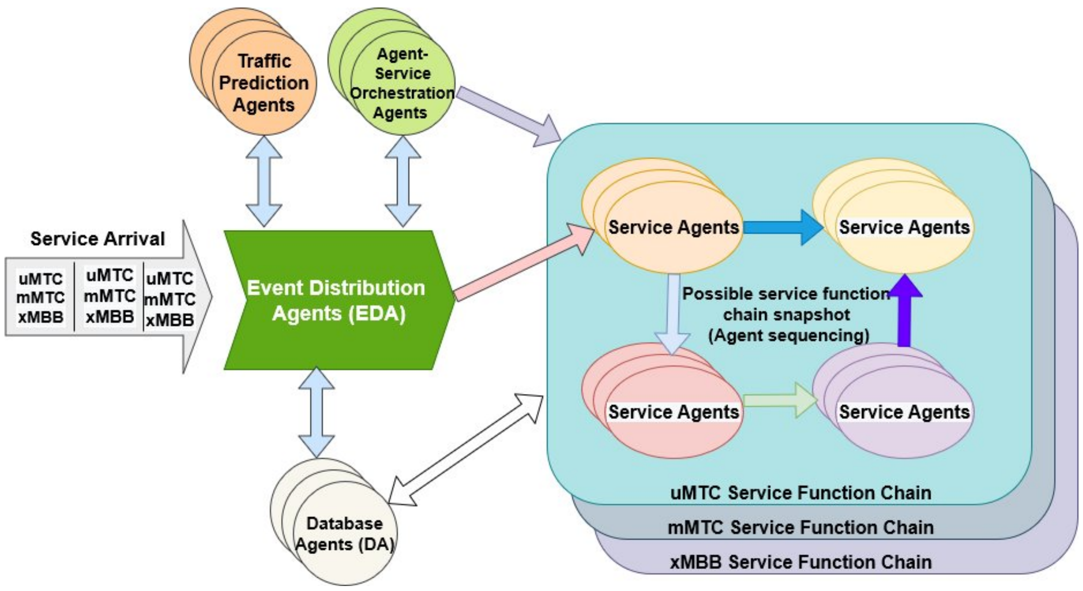

Traffic Prediction Agents
Agent-Service Orchestration Agents
Service Arrival
uMTC mMTC xMBB
uMTC mMTC xMBB
uMTC mMTC xMBB
Event Distribution Agents (EDA)
Service Agents
Service Agents
Possible service function chain snapshot (Agent sequencing)
Service Agents
Service Agents
uMTC Service Function Chain
mMTC Service Function Chain
xMBB Service Function Chain
Database Agents (DA)

FIGURE 3.10: Agent Sequencing and Scheduling for Arrival Service Processing

chain that we assumed is two serially connected agents working in parallel with another two serially connected agents, see Figure 3.10 for agents sequencing (SFC).

Based on agents' general specification guidelines presented in Section 3.6, let us provide the specification of agents to be used in the evaluation. Agents have a unique identifier number, service capacity, service state (information or facts), adaptive serving methods (mostly a machine learning model), strategy (how to perform service processing), validation/verification techniques (optional), execution(output), which are provided at the agent instantiation stage. The simulation only considered ID, capacity, type, and status (busy, idle, or fail). In the agent specification Table 3.2, the agent IDs are given as Ag_srvm_x and Ag_bp_x. Where srvm indicates service processing type of agents, bp indicates background processing type of agents and X, which could take 1, 2, 3...N, indicates number used to identify a specific agent instance from a given category. Table 3.3 presents all type of agents used in developing the overall simulation system, with detailed parameters. The table also shows different type of agents are instantiated for each type of services. These agents are use in creating SFC. Using the type of agents indicated in Table 3.3, a simulation of an autonomic network management system is developed.

The network topology considered in the simulation is represented in Figure 3.11. The figure shows a California Research and Education Network (CalREN), to be used as a backhaul or metro network in the evaluation which aggregate traffics to an edge data center. The nodes in the topology are forwarding SDN switches. Aggregate traffic containing multiple types of services such as uMTC, mMTC, and xMBB. The weights on the links and nodes are snapshot values indicating the cost of each element.

In the evaluation, the fraction of service composition is assumed to be 10:20:70 for uMTC, mMTC, and xMBB respectively. This is because there is no representative data to infer the 5G traffic composition. Table 3.4 shows the service ratio and measurement values used in the evaluation.

The overall simulation parametric values are presented in Table 3.5.

It should be noted that there are important assumptions made in the evaluation. We assumed equal computational overheads for all service processing types of

Agent specification	Service Processing Agents	Background Processing Agents
Agent ID	Ag_srvm_x	Ag_BP_x
Input	Arrived services	Change in resource/network states
Facts	Service specifications and processing demands	Resources specification and service agents states etc
Cognition	Service identification, and classification	Service workload prediction and monitoring status change etc
Plan	Mechanism of authentication, admission, path computing, routing and monitoring etc	Adaptive resource (de)allocation, seating up of SFC etc
Verification and validation	Check the computed path for consistence, optimality and validity in time and state	Verify resource state before update
Output/Action	authenticating, admitting, and routing of service in a given path or the service chain	(de)allocate resources and resource state database

TABLE 3.2: Internal-Functional Specification of Agents

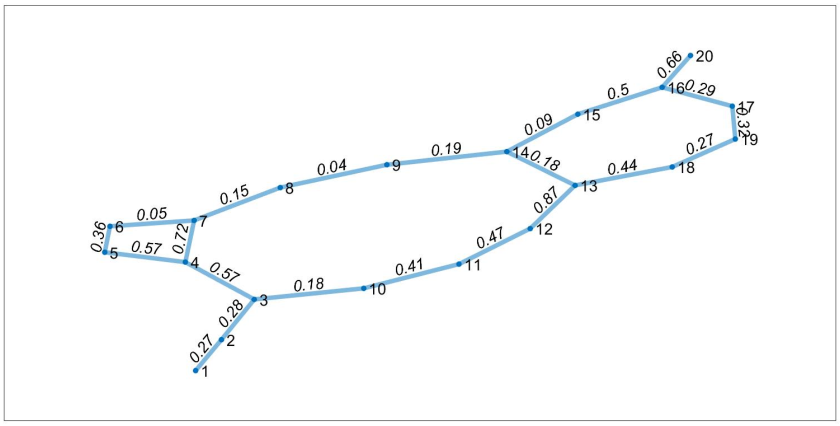


FIGURE 3.11: Considered Network Topology For Path Computation

Agent type	Number of Agents Required	Capacity(Gbps)	Processing Latency (ms)
uMTC Service Agents	depends on the incoming service workload amount	100	Service-workload/Agent-Capacity
mMTC Service Agents	depends on the incoming service workload amount	100	Service-Workload/Agent-Capacity
xMBB Service Agents	depends on the incoming service workload amount	100	Service-workload/Agent-Capacity
Traffic Prediction Agents	3 (fixed at the start of the simulation)	100	20
Even distribution Agents	3 (fixed at the start of the simulation)	100	0.03
Orchestration Agents	3 (fixed at the start of the simulation)	100	0.03
Database management agents	3 Agents (fixed at the start of the simulation)	100	0.03

TABLE 3.3: Detailed Agents Parametric-Specifications Used in Developing the overall Simulation System

Services type	Service workload proportion	Expected delay in the edge data center
uMTC (Type 1)	10%	0-2ms
mMTC (Type 3)	20%	2-4ms
xMBB (Type 2)	70%	4-10ms

TABLE 3.4: Service Specification

System Specification	System value
Queue	one service queue for each service type
CPU	10,000,000Tbps
Simulation time	120 hours (5 days of a week)
Service arrival process	Random poisson arrival
Service arrival	3 database agents (db agents)

TABLE 3.5: Simulation Parameters

agents, which may not always be true. It is because agents could perform different computations for each service type. The internal components are designed for the intended function specialization of that agents. Agents could have event monitoring, cognition, planning, verification, and action performing components to make autonomic decisions and dynamic responses. These components require CPU processing and processing delay, which can be added to the virtualization overheads (assuming agents are normally instantiated in a given container or virtualized environment).

Other agents' computational overhead is considered variable, depending on the type of agents. For instance, traffic prediction agents are expected to have different internal components than database management agents service orchestration agents. Therefore, in the evaluation, a given CPU computing overhead is assumed to be added as a computational processing workload. The detailed study of agent computational overhead is left as future work.

In general, the evaluation is based on service workload prediction, path computation, resource allocation, service workload scheduling, service processing, and network and resource state updates. Service arrivals are considered as input to the system and handling agents. The even handling agents may request the orchestration agent to instantiate and create a chain of agents for the services to be processed. The chain of agents represents a service function chain required by the services. This in fact is set proactively every hour by predicting the incoming service type and amount of service workload.

Given service progress through the sequence of service processing agents getting the necessary service treatments; it could be service authentication and authorization to use a given set of resources, SFC path, and service management. A snapshot of agents sequencing from a different type of service agents is provided in Figure 3.10. In a given type of agents, service agents are scheduled in a round-robin fashion. That means the first arrived service is scheduled to be served by the first agent from a given set of agents of that type, the second arriving services by the second agent if the first agent is still busy, and so on. The evaluation is considered to show agents' interaction to perform a given service, service schedule, and processing.

Arriving services are classified into three categories. And each service is scheduled and stored into three separate queues depending on its service categories. These are uMTC, mMTC, and xMBB queues. The scheduler allocates separate resources and SFCs depending on each queue state. This way services are scheduled maintaining service separation, priority, and service differentiation. The evaluation is considered to show agents' interaction to perform a given service, service schedule, and processing.

Generally, based on the above evaluation specifications and developed simulation environment (emulation of MANA-NMS), the simulation evaluation is performed for: service admission and processing, agent utilization and service processing, resource consumption, processing and queuing delay, agent and server (agent host) failure analysis, and building a resilience system and its impact on resource consumption and processing delay. We performed the simulation for 120hrs, representing a five days traffic pattern. The results are discussed in the following subsections.

3.7.2 Service Arrival and Processing Evaluation

Arriving services are allocated a network resource after the best available path is computed. The service schedule is based on service priority. It is in the order of

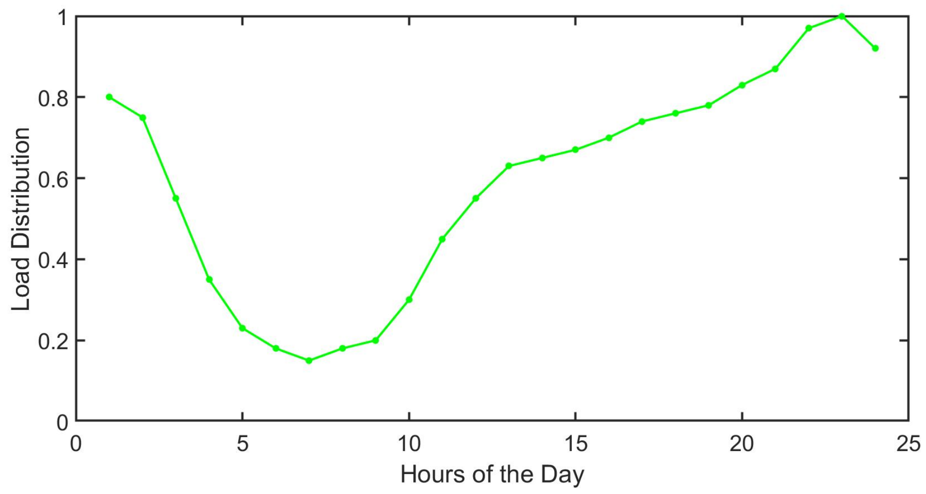


FIGURE 3.12: Workload Distribution

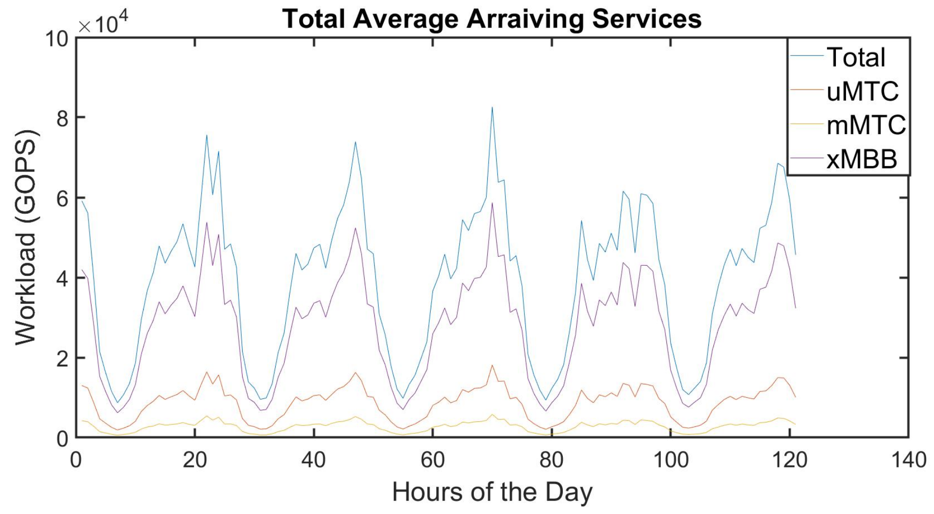


FIGURE 3.13: Average Arriving Service Workload

URLLC (high priority)–> mMTC –> xMBB (least priority) in a given service scheduling period. Based on the amount of arriving service workload, resources are allocated to agents before they are instantiated. Based on the total serving capability of agents, a service is injected using the workload percentage shown in Figure 3.12. The evaluation also considers three types of services with a different workload demand to the autonomic network management functions. The arrival is assumed to be Poisson distributed with a workload distribution, shown in Figure 3.12. The arriving service patter is plotted in Figure showing the total average service workload for each type of service and the overall services workload arriving to the edge data center. Based on the availability of service agents, services are scheduled until all active agents are fully loaded. Figure 3.13 shows average arriving service workload over a 120hrs evaluation period. Figure 3.14 shows rejection of services due to agents overload. The overall service admission probability of the multi-agent system is calculated as the ratio of service admitted to service requested. The result is presented in Figure 3.15. As it can be seen from the Figure, service admission is approximately one during non-peak hours and whereas there is some probability of services rejected at peak hours.

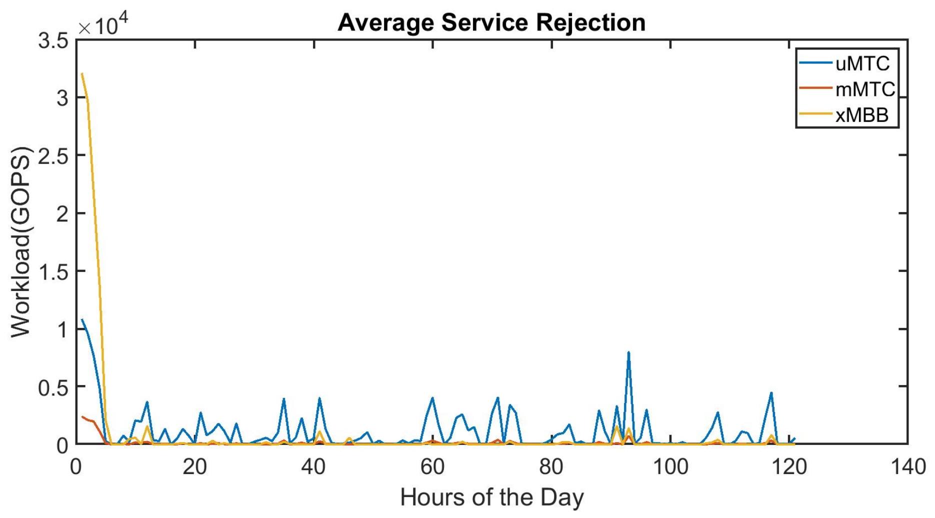


FIGURE 3.14: Service Rejection

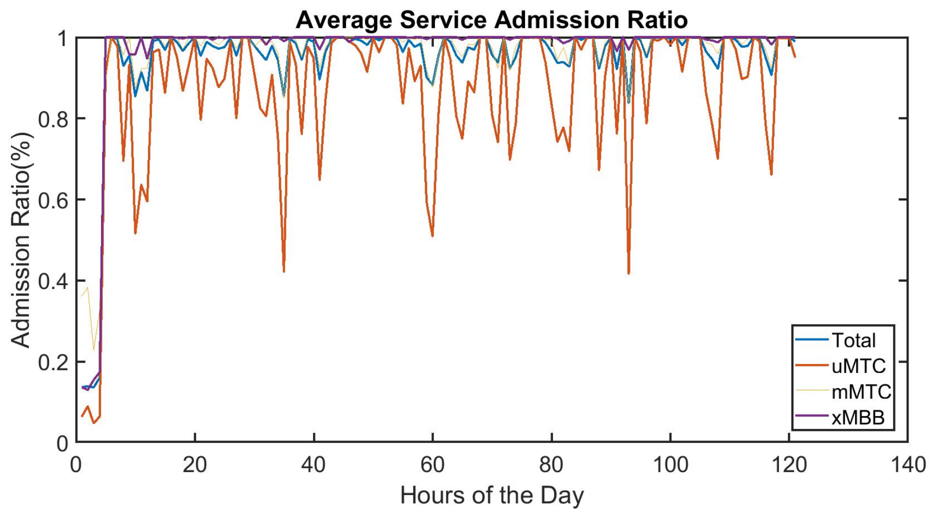


FIGURE 3.15: Service Admission Probability

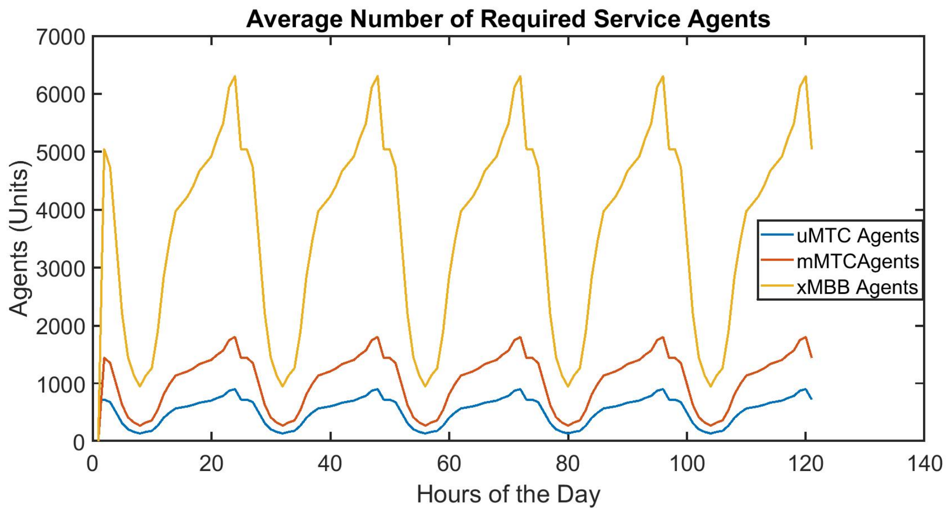


FIGURE 3.16: Average Number of Required Service Agents

3.7.3 Agent Utilization and Edge Data Center Resource Consumption

Agents require CPU resources to process services. The instantiating of multiple agents consumes data center resources such as memory, storage, CPU, and network bandwidth. In our simulation, we considered only CPU, assuming equivalent requirements would be scaled for memory, storage, and bandwidth. As indicated in Table 3.3, there is a fixed number of background processing agents. They are instantiated at the beginning of the simulation. These agents are even handling agents, service workload prediction agents, orchestration agents, and network and resource state database management agents. There are also service agents that are scaled according to the incoming service workload demands. In our simulation, we considered only the number of agents that are scaling with the workload. However, it is also possible that a given agent could scale its processing capability according to the service demands. This could be possible by dynamically adding the necessary resource to the agent. Such kind of agent's capacity scaling is called "horizontal scaling". However, the discussion of such a scenario is out of the scope of this work.

The number of active agents instantiated at a given time is directly related to the service prediction. Service workload is predicted by the service workload predicting agents. On the other hand, the CPU consumption in the edge data center is directly related to the number of active agents instantiated at a given time. Since active agents consume resources, it is necessary to reduce the number of active (but idle) agents to have efficient resource utilization. Figure 3.16 depicts the average number of required agents in a given hour. This is calculated based on the predicted arriving service workload.

Agents are assigned to provide service scheduling and monitoring functions. Once agents are assigned to accomplish the given task, they will be locked into a busy (internal execution) state before becoming the idle state. The average number of busy agents and idle agent during the 120hrs simulation period are depicted in Figure 3.18 and Figure 3.17. The Figures indicate the average number of agents that are actively performing service processing and waiting for service scheduling, respectively. The idle number of agents is a total average over the whole period which is approximately 4000 agents. This number indicates idle agents that are actively consuming resources without providing services. This shows the inefficiency of resource consumption which is the consequence of the prediction model accuracy.

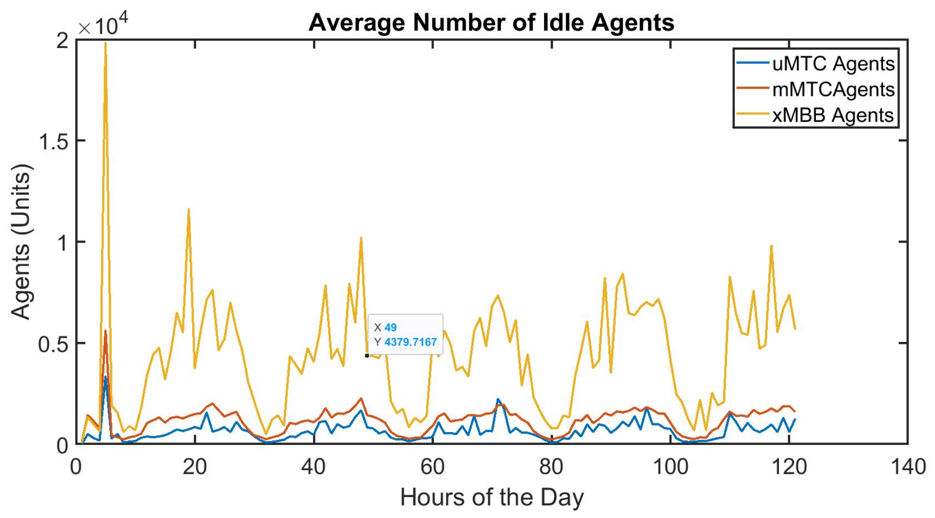


FIGURE 3.17: Average Number of Idle Agents

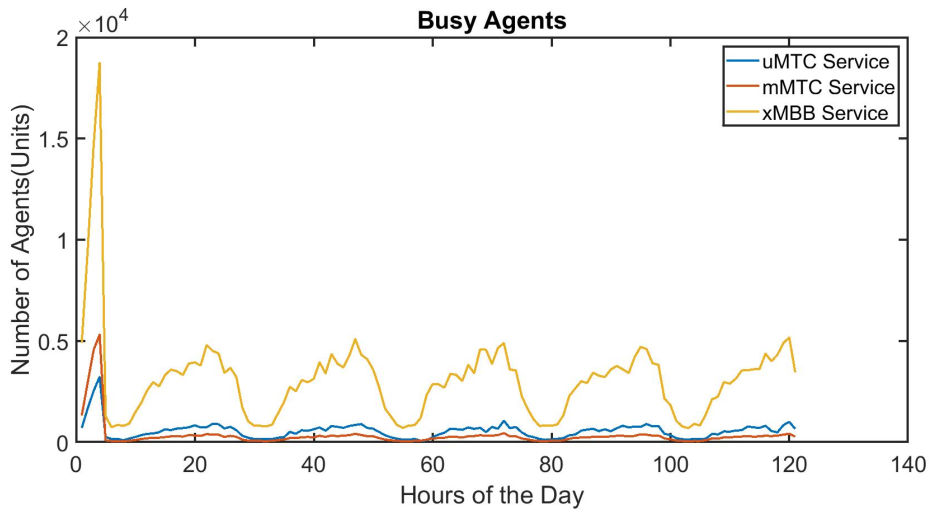


FIGURE 3.18: Average Number of Busy Agents

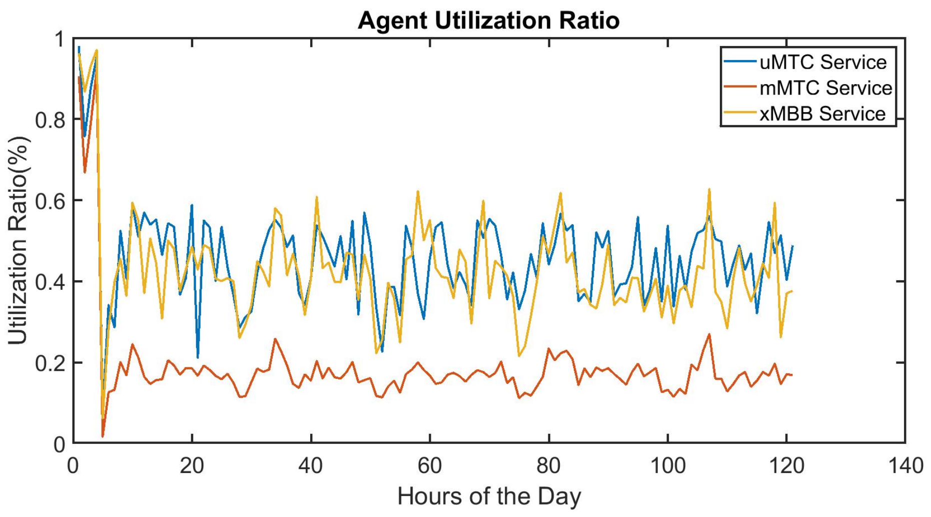


FIGURE 3.19: Agents Utilization

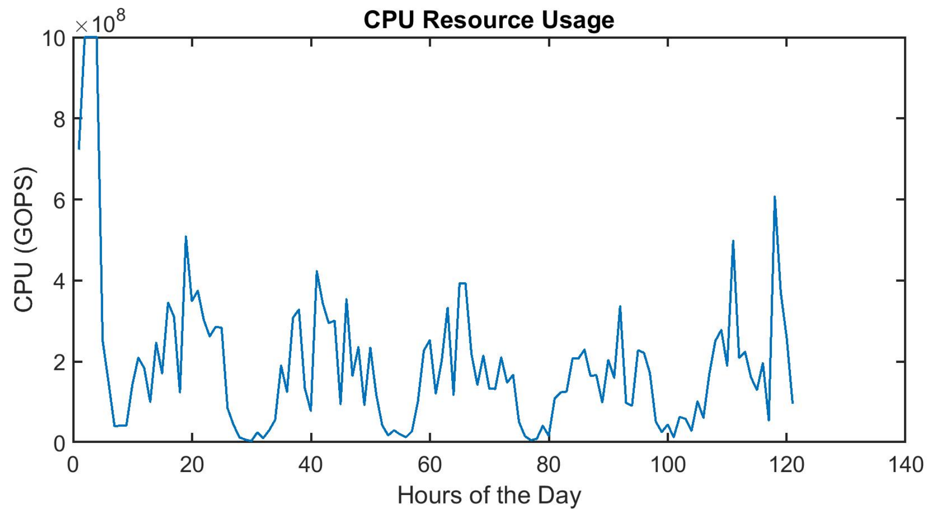


FIGURE 3.20: Edge Data Center Resource Utilization

However, if we reduce these idle agents, we may increase the service rejection and service latency in the edge data center. This is because the service arrival is a random process and an exact prediction is dependent on the prediction model used in the prediction agents. Moreover, given the model used in the prediction agent, the prediction is the optimal value to avoid further service rejection and service delay in the data center. The overall service-processing-agents utilization by each type of services is depicted in Figure 3.19.

Figure 3.20 shows the average amount of resource consumption in the edge data center in terms of the consumed CPU. The maximum available CPU resource in GOPS is given in Table 3.5. As can be seen from the Figure, the resource consumption is directly related to the number of total active agents which is also directly related to the service workload arrival pattern. As indicated the arrival pattern is predicted by the service workload prediction agents.

Moreover, we have evaluated agents' load ratio over 120hrs period for each service type agent. This is an average busy period for a given type of agent. The evaluation shows, on average, how much a given agent is utilized over the 120hrs period. This parameter indicates the efficiency of agents in processing and delivering the

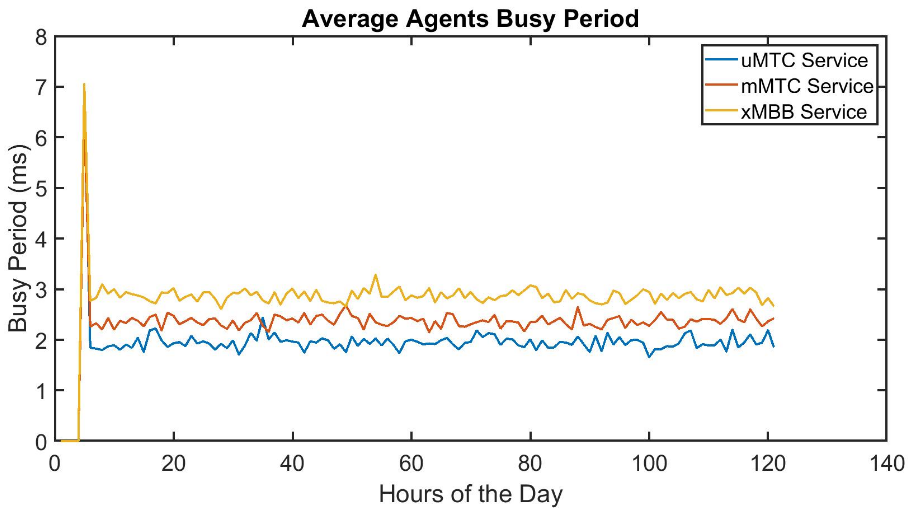


FIGURE 3.21: Agent Busy Period

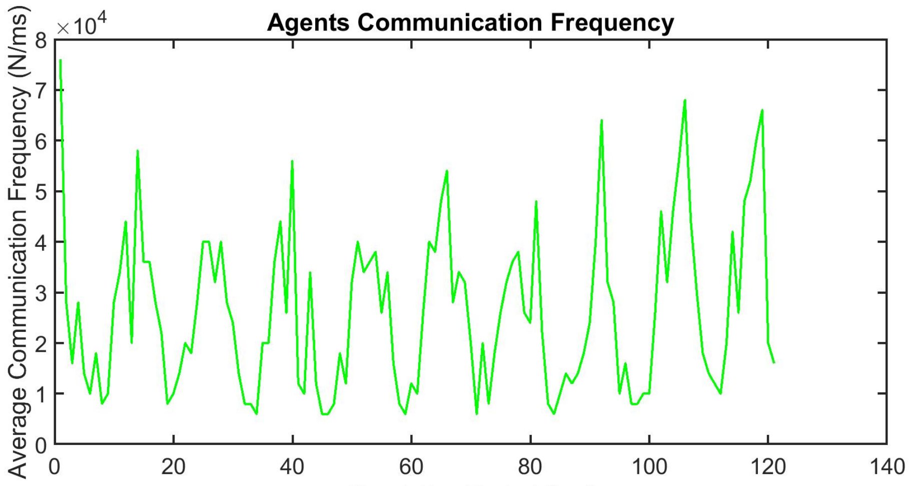


FIGURE 3.22: Agents Average Communication Frequency for Database Update due to Service Arrival

given service while the agents are in an active state.

One capability of an autonomic multi-agents network management system is the ability to communicate between agents to accomplish a given task. In our evaluation scenario, we considered agent communication between scheduling and monitoring agents and resource database management agents. Accordingly, the evaluation indicated the communication frequency needed to accomplish the given arriving service workload. Figure 3.22 shows the average communication frequency over a 120hr period.

3.7.4 Service Queuing and Processing Latency Evaluation

The simulation has two main stages of queuing for each type of service. These are service a arrival queue and a service processing queue. As the services arrive at the edge data center, they are first stored in the service arrival queue of the corresponding service queue before they are scheduled for processing in the service agents.

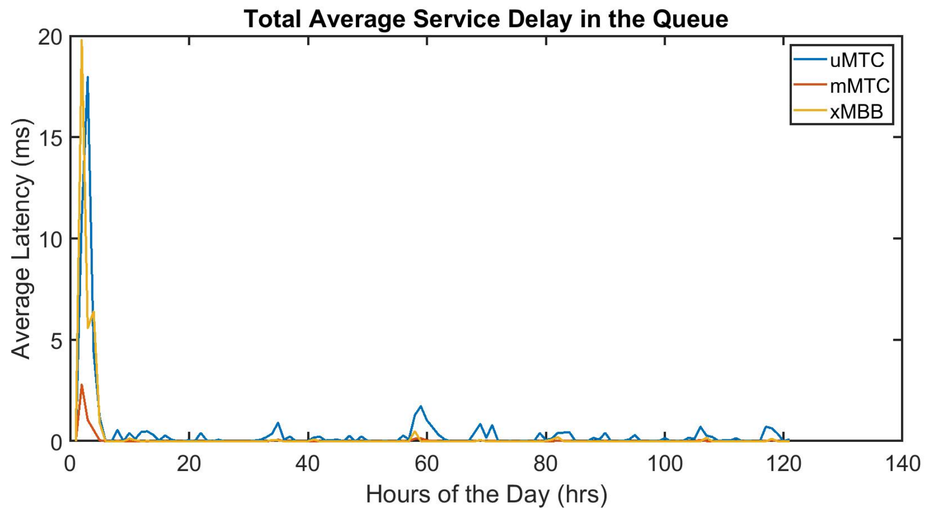


FIGURE 3.23: Arrival Service Queuing Delay

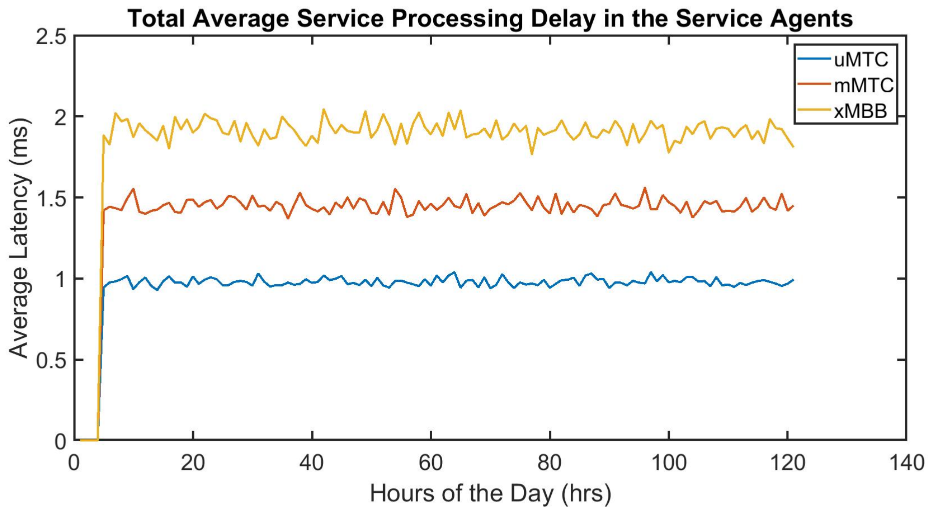


FIGURE 3.24: Service Processing Delay in the Service Processing Agents

Service is processed sequentially through the service function chain. Since we considered a maximum of two agents connected serially, there is a service processing queue between the two serial connected service agents. In this scenario, the service is scheduled and processed in the service function chain which is connecting service agents. The simulation result depicted in Figure 3.23 presents the average delay in the queues for each type of service. Figure 3.24 depicts the average processing delay in service agents passing through the service function chain. The total delay in the edge data center is assumed to be the sum of the two delays as formulated in the above section. Figure 3.25 shows the total end-to-end latency in the edge data center which ignores the communication link delay between subsequent service agent delay in the SFC.

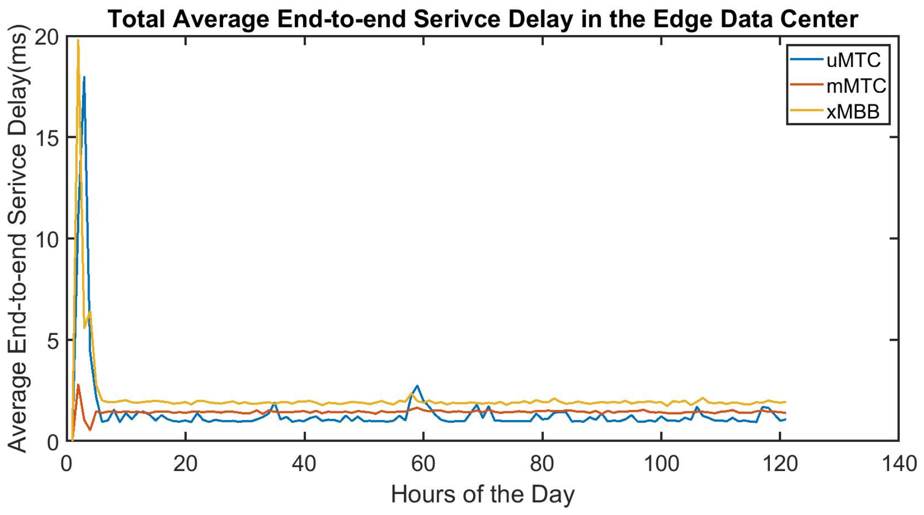


FIGURE 3.25: End-to-End Service Latency in Edge Data Center

3.7.5 System Evaluation for Fail-over Scenarios

In the above evaluation scenarios, agents are assumed to be 100% reliable and available. However, this may not be true as agents are software components with a non-zero probability of failure. The agents are an event handler, orchestration, prediction agent, and database management. These agents are very critical for system resilience. Therefore, such agents are redundant with a well-synchronized backup system. Such types of agents are fixed in our simulation, and a few are compared to service processing types of agents. And assumed to be 100% reliable. However, service processing agents are assumed to be less reliable and their number is scaled according to the arrival service workload. In the simulated system, such types of agents are not protected by a backup system. However, overpraising of 1% is considered to accommodate the imperfection of the service workload prediction, which is dependent on the machine learning model incorporated in the service workload prediction agents.

Different failure scenarios could be analyzed using the developed simulation environment. It could be a backhaul network node failure that affects the link utilization and service admission or server/agent failure which affects the edge data center service admission probability and service latency. A network node failure is handled by recomputing new available paths until the node is back into operation. This is a classical scenario thoroughly considered in the literature. We consider limited scenarios for the multi-agent-based autonomic network management system failure. In simulation two fail-over scenarios are considered; a single or a very few numbers of agents failure and single server failure. A single agent failure has an insignificant impact on the overall service admission and service delay in the edge data center. This is because agents are lightweight service units that can be instantiated easily and in a few more for overprovisioning, with little increase in resource consumption. They can also be easily and quickly instantiated to compensate for the reduction in the number of service agents due to service agent failure.

However, the failure of a server, hosting several agents, could significantly affect the edge data center processing capacity. The extent of the impacts depends on how the SFC is created. That means if a single type of agent is hosted in a single server, and if that server fails, the impact will be the highest. It is because a given service may need to get service treatment from all types of service processing agents in the

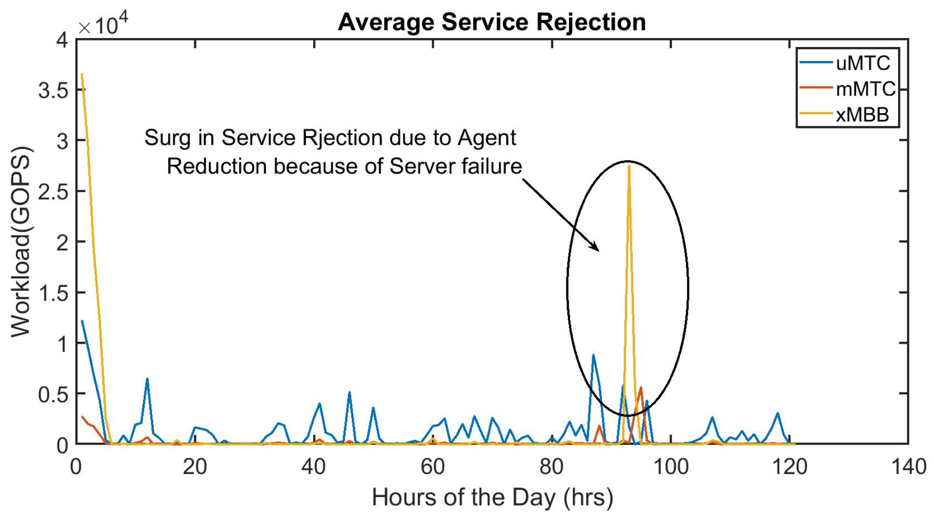


FIGURE 3.26: Impact of Server (Hosting Service Processing Agents) Failure on Service Rejection

SFC. The failure of such server would significantly reduce the number of agents of a given type, creating a shortage of such agents for SFC creation. The least impact could be achieved by rightly balancing and distributing the agent type to different servers. With such considerations, the simulation is performed for maximum impact of hosting a single type of agent in a single server. Moreover, a single server failure at a random time within the 120hrs simulation period is introduced. And the impact on service admission probability and service processing latency is evaluated.

As it can be seen from Figure 3.26, the impact of a server failure is significant in that it increased the service rejection. However, the impact does not last long because the agent monitoring and orchestrating other agents immediately detects the reduction in the number of service processing, agents and next instantiate additional agents of such type. The instantiated agents are updated with the service state and information to take over the service processing. After instantiating the required service processing agents, as a replacement to the failed agents, the orchestrating agents also create the necessary SFC. Figure 3.27 depicts the overall service latency in the edge data center. As can be seen from the Figure, there is a significant latency on each type of service processing, depending on the time of failure of the server hosting a given type of service processing agents. Figure 3.28 shows the impact of server failure on the service admission probability. As it can be observer from the Figure, there are significant reduction in the admission probability for all type of services,depending on the time of failure of server hosting a given type of service processing agents.

3.7.6 System Reliability Evaluation and Service Differentiation for Resource Efficiency

Reliability of network management system is very important. In Section 3.6, we discussed and presented a reliability improvement technique based on a backup system. We evaluate the system performance by introducing backup service processing agents. Figure 3.30 shows the admission probability using backup system. Compared to the result obtained without backup, which is depicted in Figure 3.15, the admission probability has increased. Moreover, Figure 3.29 shows the reduction of

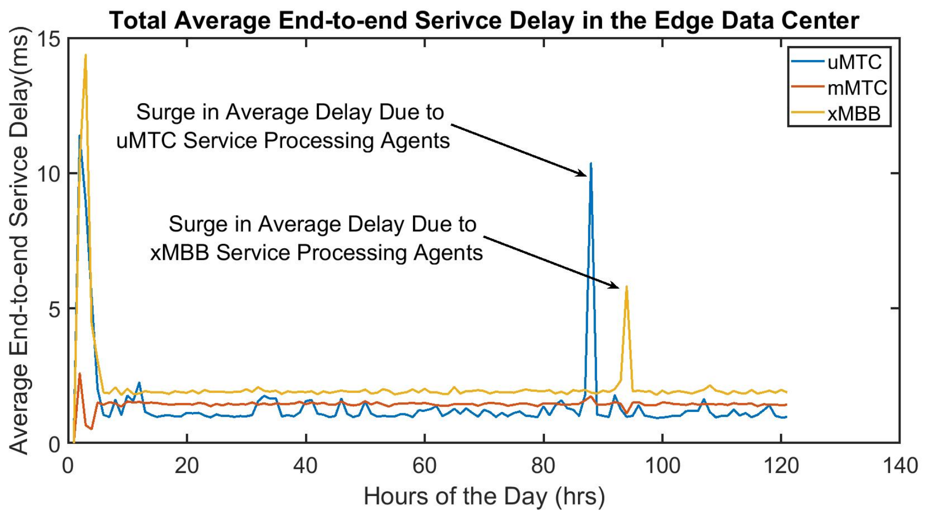


FIGURE 3.27: Impact of Server Failure on End-to-End Latency in the Edge Data Center

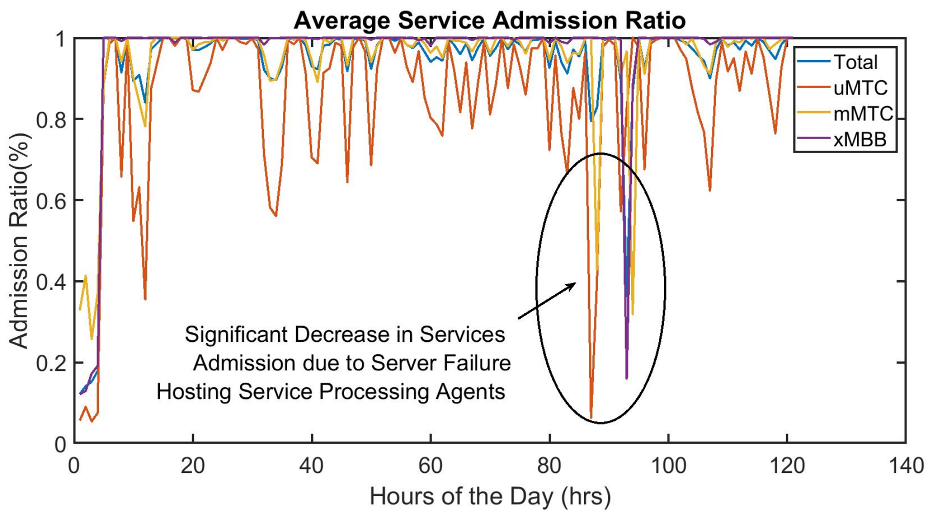


FIGURE 3.28: Impact of Server Failure on Service Admission Probability

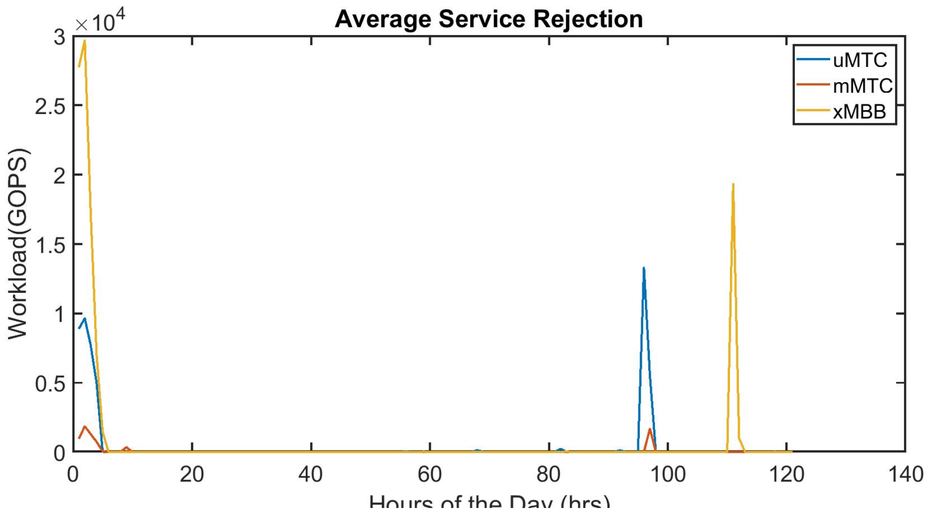


FIGURE 3.29: Service Rejection with Backup Service Processing Agents

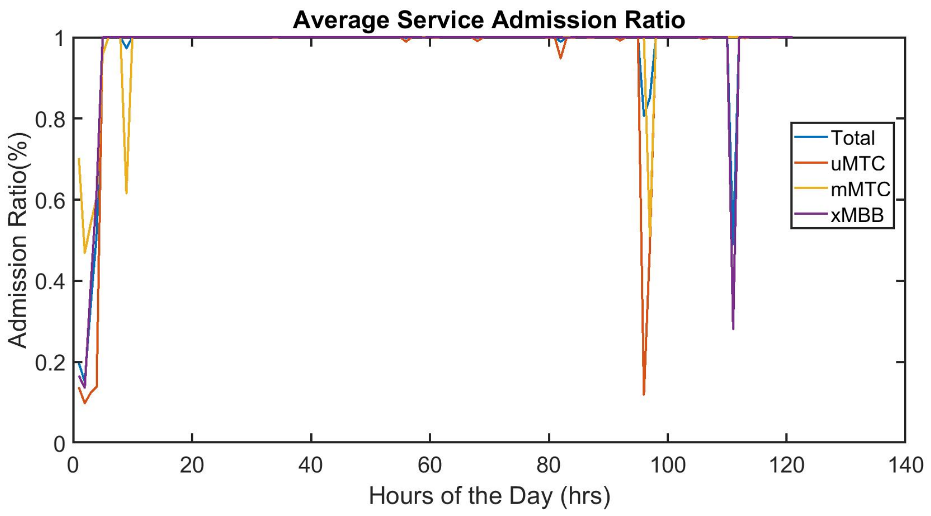


FIGURE 3.30: Service Admission Probability

service rejection compared to the system without back service processing agents, see Figure 3.29 and Figure 3.14.

The spike (seen in the figure) in-service rejection is due to the introduced server failure impacting the available service processing agents. This shows that even if we have a backup system, the impact is significant for a small duration. This is because, after the active serving agents' unexpected sudden failure, it needs time until new service agents are instantiated and taking over the service processing abandoned by the failed agents. Figure 3.30 shows the admission probability improvement by using backup system.

Figure 3.31 shows the increase in resource consumption compared to Figure 3.20, which is plotted for the system without backup service processing agents. Since there is a large number of service processing agents, having a backup for all of them approximately doubles the resource consumption. This indicated that a backup system has a huge cost of resources and energy consumption. To reduce such consumption, we propose service differentiation in providing a backup system by providing backup agents for only critical services or by providing flexibility to provide such

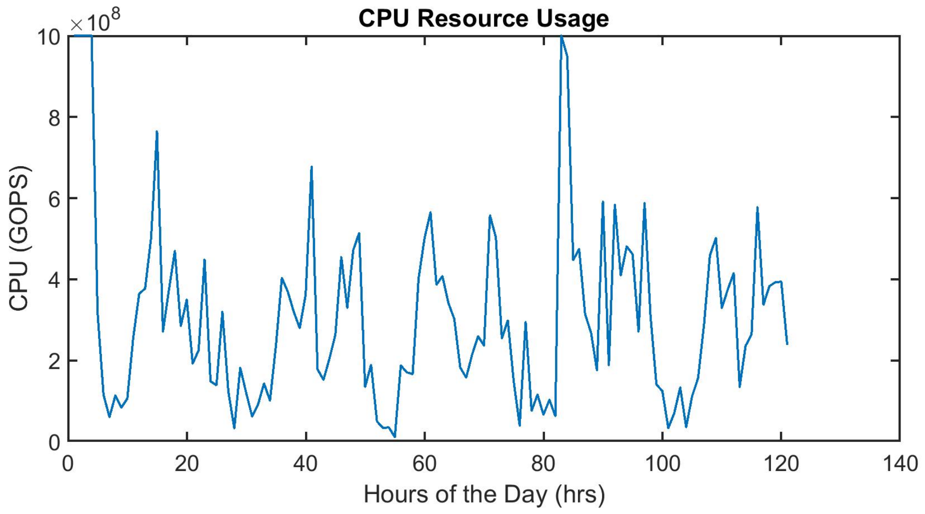


FIGURE 3.31: CPU Resource Consumption with Backup System

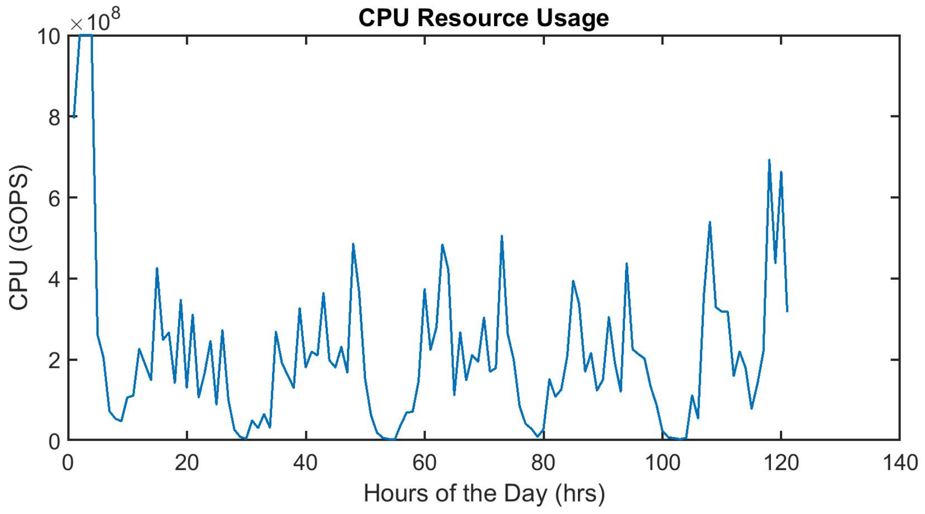


FIGURE 3.32: CPU Consumption with Backup System for uMTC Service Processing Agents

options to use by the network operator to decide. This makes the system more efficient in resource utilization and resilient compared to a system without any backup mechanism. A more detailed analysis is left for future work. Figure 3.32 shows the reduction in CPU consumption compared to a backup system without service differentiation. However, more critical services are admitted with a few increase in resource consumption compared to the system without service backup.

3.8 Autonomous Network Traffic Classifier Agent for Autonomic Network Management System

3.8.1 Overview and Background

In this section, we give an overview of the technologies and concepts that have been used in this paper. As technology evolves, complex problem solving is also evolving from centralized to distributed solutions. Distributed artificial intelligence (DAI) is a solution where many entities work together to solve a complex problem [81,

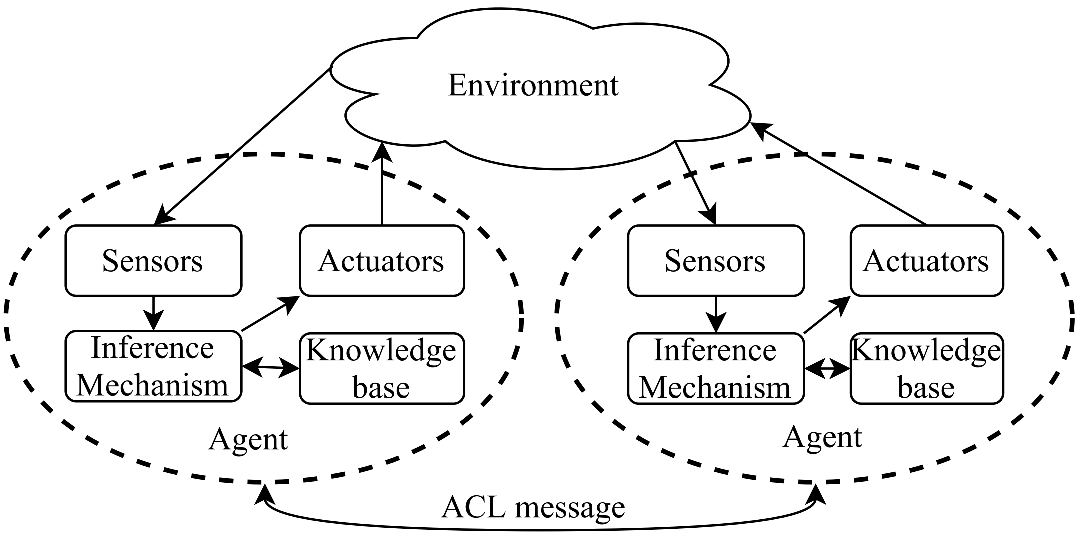


FIGURE 3.33: General overview of a MAS.

244]. In DAI, historically, there are two sub-regions. The first is distributed problem solving, which focuses on breaking down a problem and using slave nodes to fix it. The micro solutions are then gathered and reconstructed to produce the actual solution. The second sub-area is MAS that emphasizes agents with a certain degree of autonomy and uncertainty, resulting from their interactions in their joint activity.

A MAS is made of autonomic entities referred to as agents as illustrated in Fig. 3.33. Similar to distributed problem solving, the agents jointly work to solve tasks in a more flexible manner since they are also capable of making autonomous decisions. These agents use information obtained through interacting with other agents and the environment to learn new knowledge and perform actions to solve given tasks accordingly [80]. MASs are characterized by some features such as Autonomy: agents function without human or other direct intervention and have some form of influence over their behavior and inner state, Social ability: agents communicate through some kind of agent-communication language (ACL) with other agents, Reactivity: agents perceive and react promptly to changes that occur in their environment (which could be the physical world, a user via a graphical user interface, a group of other agents, the Internet, or maybe all of these combined), Pro-activity: agents are not merely behaving in reaction to their environment; by taking the initiative, they can display goal-directed actions.

3.8.2 Machine Learning Models For Network Traffic Classification

Machine learning (ML) is a paradigm that focuses on giving systems the capability to learn from experience automatically without being programmed explicitly [140, 159]. ML facilitates the analysis of vast volumes of data. While it typically offers quicker, more specific outcomes to identify lucrative possibilities or hazardous threats, it may also take additional time and resources to properly train it. To search for patterns and trends in data and make informed choices in the future based on the training examples, the learning process has to start with some training observations or data. This could be from examples, direct experience, or guidance, from which a model is developed based on the learned patterns and trends. The primary objective is to allow computers to learn automatically and change behavior accordingly, without human intervention. There are several ML methods e.g. supervised learning,

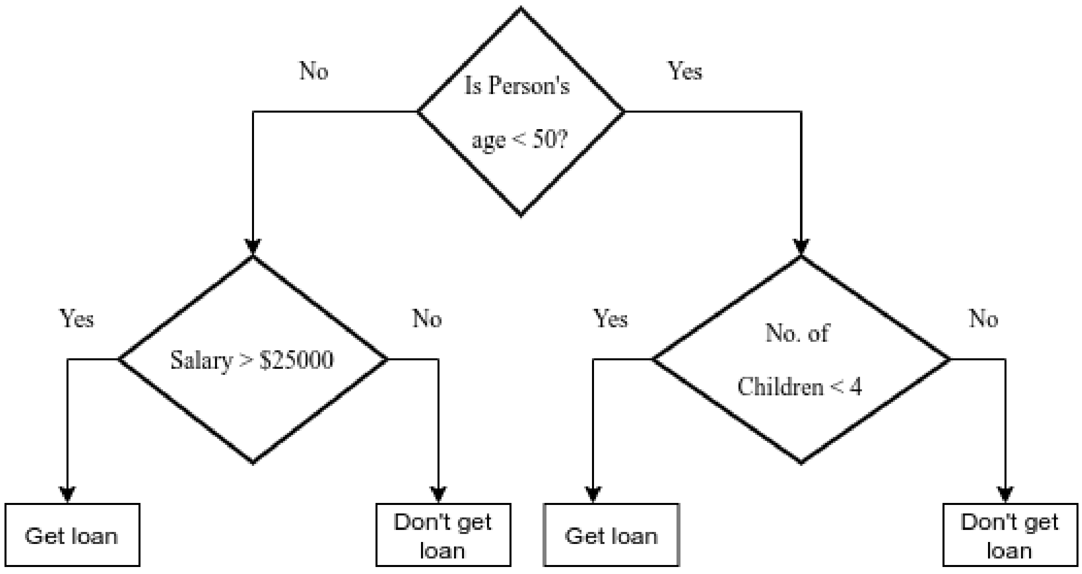


FIGURE 3.34: An Example of a Decision Tree.

unsupervised learning, semi-supervised learning, and reinforced learning [198]. In this work, supervised learning methods are considered.

Supervised learning algorithms apply past knowledge, learned from a training dataset to predict future events and outcomes. In the training dataset, there are labeled input-output pairs. The learning algorithm produces an inferred function to make predictions about the output values, starting from the analysis of a known training dataset. After ample preparation, the system can provide targets for any fresh data. To adjust the model accordingly, the learning algorithm can also compare its output with the right, expected output and find errors.

Decision Tree Classifier Algorithm

A non-parametric supervised learning approach, used for classification and regression is the decision trees. Decision trees learn from knowledge with a set of if-then-else decision rules to approximate a sine curve. The deeper the tree, the more complicated the laws of decision are and the fitter the model [43]. The decision tree helps to develop classification or regression models in the context of a tree structure.

A dataset is broken down into smaller subsets, while an associated decision tree is progressively built at the same time. A tree with decision nodes and leaf nodes is the outcome. There are two or more divisions of a decision node. A classification or decision defines the leaf node. In a tree, the top decision node corresponds to the best predictor called the root node. Both categorical and numerical data can be treated by decision trees.

Naive Bayes Classifier Algorithm

It is a classification method based on Bayes' Theorem and the assumption of predictor independence. A Naive Bayes classifier, in simple terms, assumes that the existence of one feature in a class is unrelated to the presence of any other feature [43]. A football, for example, is a rigid, circular object with a diameter of about 9 inches. Even if these qualities are dependent on one another or on the existence of

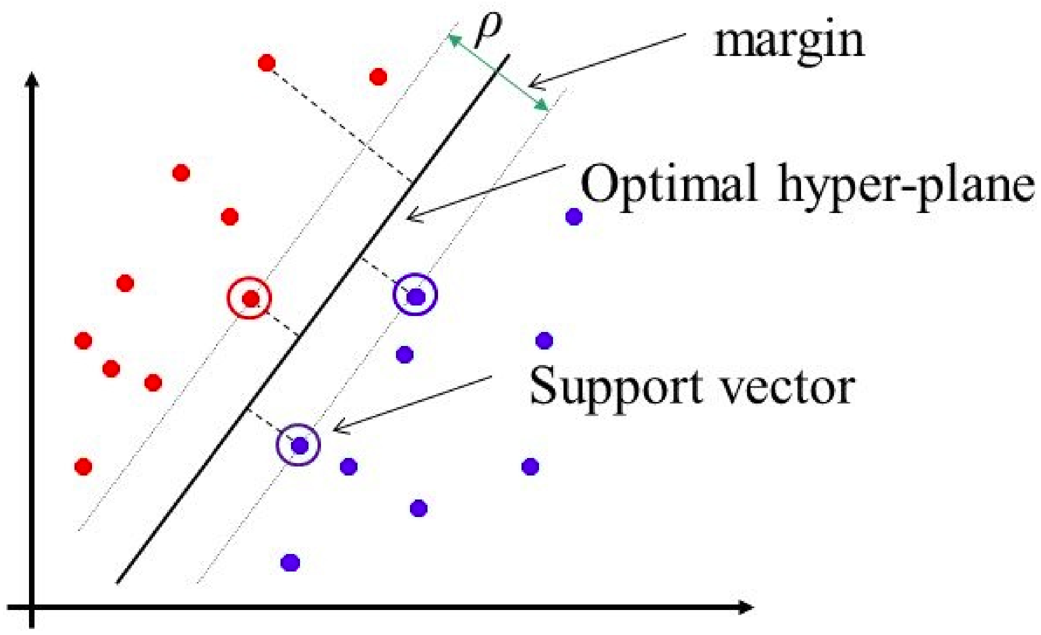


FIGURE 3.35: An Illustration of a Hyperplane and Support Vectors.

other qualities, they all lead to the probability that this object is a football, which is why it is called "Naive."

The Bayes theorem allows you to calculate posterior likelihood of class (c, target) given predictor (x, features) $P(c|x)$ from prior probability of class $P(c)$, prior probability of predictor $P(x)$, and the likelihood $P(x|c)$ using $P(c)$, $P(x)$, and $P(x|c)$ like in the following equation:

$$P(c|x) = \frac{P(c) * P(x|c)}{P(x)}$$

The Naive Bayes model is easy to develop and is especially useful when working with large data sets. Naive Bayes is considered to outperform even the most advanced classification methods due to its simplicity[43].

Support Vector Machine Classifier Algorithm

SVM algorithm is to find a hyperplane that separately classifies the data points in N-dimensional space (N is the number of features). There are a lot of potential hyperplanes that could be selected to distinguish any two groups of data points. The goal is to find a plane with the highest margin, i.e. the maximum distance between both classes' data points. Maximizing the gap from the margin offers some support so that further confidence can be classified in future data points [43].

Figure 3.35 illustrates hyperplanes, support vectors and margins in relation to SVM [264]. Hyperplanes are boundaries of judgment that help to classify the data points. It is possible to attribute data points falling on either side of the hyperplane to various groups. Support vectors are data points that are closer to the hyperplane and affect the hyperplane's direction and orientation. We optimize the margin of the

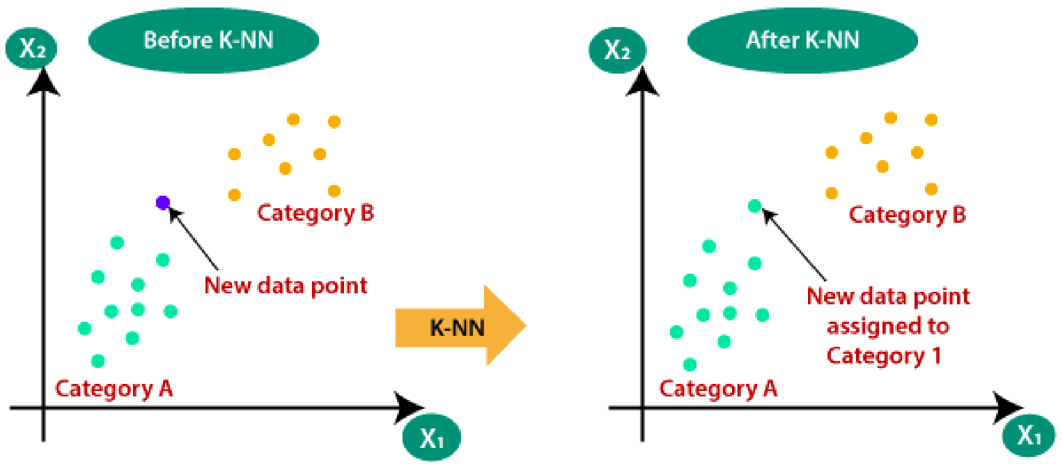


FIGURE 3.36: An Illustration of the K-NN Classifier Algorithm in action.

classifier using these support vectors. Deleting the support vectors would change the hyperplane's location. These are the points that are helping to build the SVM.

K-Nearest Neighbours Classifier Algorithm

K-NN classifier [43] works by storing the different instances of the data in the training dataset. The classification here is solely based on a majority vote of all the neighbors closest to the data point in question. For example, the data point in question is assigned to a class that has the most data representatives within the neighbors nearest to this point.

Given two categories of data, i.e., Category A and Category B, and a new data point x. The question at hand is: in which of the two categories does the data point lie? To solve this, the K-NN classifier algorithm comes in handy. The working principle behind the K-NN algorithm is explained below.

1. In this first step of K-NN, the data set (training and test data) is loaded.
2. A choice of the value of K i.e. the nearest data points is made where K is an integer.
3. For each sample in the test data, calculate the distance between test data and each row of training data with the help of any of the methods namely: Euclidean.
4. Take the K nearest neighbors as per the calculated Euclidean distance.
5. Among the K neighbors, count the number of the data points that belong to each class. The class assigned to the test point is based on the most frequent class.

This method helps a great deal when large training datasets are encountered. The major challenge is the determination of the value of K. One more challenge associated with this algorithm is the high computational cost since the distance parameter has to be calculated for each sample in the data set. The ambiguity with "distance" as a parameter is another challenge.

3.8.3 Proposed Network Traffic Classifier Agent

In this section, we present the architecture for the proposed NTCA. Let us first define and contextualize agents.

Agents

Agents have been defined in different ways based on different fields of study. However, the idea of agents is a generic one and can be applied across several disciplines [80]. Therefore, a more generalized definition is presented, considering the agent's abilities and characteristics. An agent is an object that is put in an environment and senses the various parameters used to make a decision based on the object's target goals or objectives. Based on this decision, the agent carries out an appropriate action. This definition is made of four keywords. Object refers to the kind of agent. It could be a software agent, e.g. network security agents, or a hardware agent, e.g. a robot.

Environment is a location in which the agent is placed. The agent obtains some information about the environment that guides the decision-making process. Parameters refer to the different forms of data collected from the environment by the agents. For example, the parameters of an archaeological robot agent are material of underground substance and depth from ground level. The agent takes actions that influence changes in the environment in which they operate. The actions may be continuous or discrete.

The key role of each agent is to solve tasks assigned to it under given constraints such as time and available resources [80]. To perform its role diligently, the agent first observes the environment for its parameters. Given these parameters, the agent then builds up a knowledge base about the environment. This agent can also use knowledge learned from other agents. This knowledge is shared among agents using an ACL as illustrated in Fig. 3.33. This knowledge base contains a history of previous actions taken. This, together with the information from the sensors, is fed into the inference mechanism, which decides the appropriate action to be taken by the agent [227].

A single agent is well equipped to work autonomously and perform actions to solve a given task. To harvest the benefits of agents, it is best to have them work together and collaborate to find solutions towards complex tasks. These multiple agents working together are known as MAS. This work contextualizes a MAS as ANMS while the agents, contained in the MAS, are the network decision elements (DEs). These agents work together while exchanging knowledge, towards a global network decision.

Network Traffic Classifier Agent Architecture

The NTCA takes in network traffic flows as input and produces at the output different classes or categories to which this traffic belongs. Network traffic classification facilitates network decisions such as resource allocation in applications such as network slicing. In order to classify traffic, the NTCA design architecture is proposed and presented in Fig. 3.37.

It is possible to specify the characteristic description and specification of the NTCAs. These NTCAs are considered as the atomic units in MANA-NMS. Agents have an input to accept a traffic classification service-request and environment measuring points. Moreover, the classifier agent is equipped with a number of facts such as

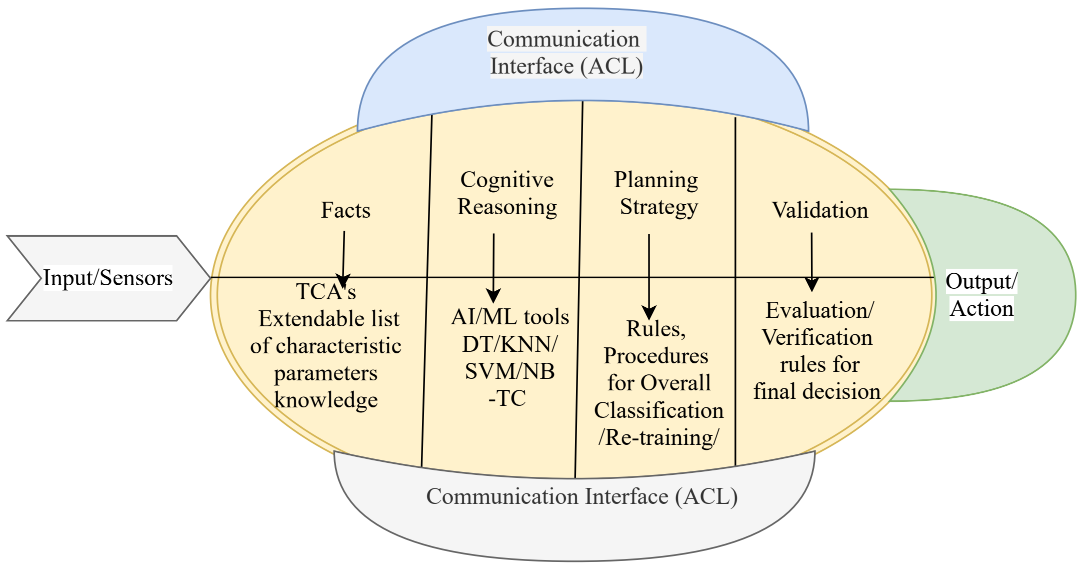

Communication Interface (ACL)
Facts
Cognitive Reasoning
Planning Strategy
Validation
Input/Sensors
TCA's Extendable list of characteristic parameters knowledge
AI/ML tools DT/KNN/ SVM/NB -TC
Rules, Procedures for Overall Classification /Re-training/
Evaluation/ Verification rules for final decision
Output/ Action
Communication Interface (ACL)

FIGURE 3.37: Network Traffic Classifier Agent Architecture

the features of the incoming traffic such as the packet size, internet protocol (IP) addresses, labels (protocols), etc. Cognition (reasoning) unit is the brain of the agent that gives it the reasoning capability. Planning strategy unit of the agent organizes the steps or procedures for the action to be taken to satisfy the requested service. The validation process comprises of some rules upon which a final decision can be made i.e. knowing the classes of traffic coming in can be very helpful in making network-related decisions. And the last component of a NTCA is an output/action unit which is the final decision (result or action) to be taken by the agent.

Communication in MANA-NMS

Agent work with other agents. In other words, the overall system such as MANA-NMS is built using the atomic agents as a building block. Since these agents are loosely coupled and could be deployed in a distributed environments, there should be a mechanism to communicate among the agents in the system. In general, communication between the agents can be classified based on the architecture of MAS and the type of information which is to be communicated between the agents. ACL is mostly used in the literature. However, other communication languages such as REpresentational State Transfer application program interface (Restful API), WebSocket, google Remote Procedure Calls (gRPC), GraphQL, etc. In this paper, we used ACL as a communication language among the agents. Evaluation of the performance of the system is dependent on the type of communication language used.

3.8.4 Performance Evaluation of Classifier Agent

The design and implementation of the NTCA is done using Python programming. osBrain, a general-purpose MAS module implemented in Python by OpenSistemas, is used to design the MANA-NMS system and test the performance of the NTCA. A number of other modules have also been considered such as pandas for data manipulation and data analysis, Numpy for scientific computing, and Scikit-learn for ML algorithms.

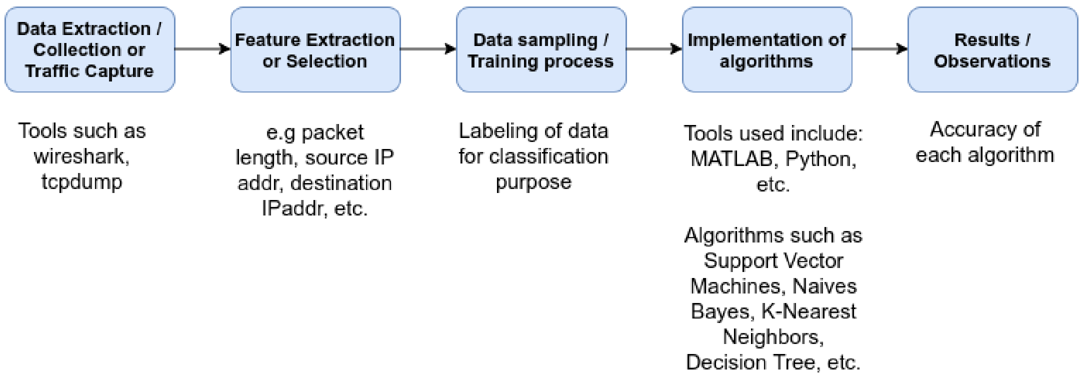

Data Extraction / Collection or Traffic Capture
Feature Extraction or Selection
Data sampling / Training process
Implementation of algorithms
Results / Observations
Tools such as wireshark, tcpdump
e.g packet length, source IP addr, destination IPaddr, etc.
Labeling of data for classification purpose
Tools used include: MATLAB, Python, etc.
Algorithms such as Support Vector Machines, Naives Bayes, K-Nearest Neighbors, Decision Tree, etc.
Accuracy of each algorithm

FIGURE 3.38: The Process of Traffic Classification.

Conceptual Framework for the Network Traffic Classifier Agent Implementation

Based on the proposed NTCA design guideline, as presented in Fig. 3.37, the agent implemented. The NTCA takes in network traffic flows as input and produces at the output different classes or categories to which the traffic belongs. Network traffic classification facilitates network decisions such as resource allocation in applications such as network slicing.

To offer autonomic service management, such as resources allocating , services scheduling, etc., it is required that all incoming network traffic be classified according to the protocol to which it refers. Protocols then relate to different applications and services. To perform network traffic classification, an agent having an ML model as an internal component is considered.

Since we assumed the network traffic to be have labels (protocols or port numbers), a supervised ML model is used in the NTCA design. Supervised learning algorithms such as K-NN, Decision Tree, SVM, and Naive Bayes are employed for performance comparison. We also assumed the agent is capable of collecting incoming traffic as a historic dataset to label it and use it to train and re-train its model. We assume the data to be stored in a network database, which we assumed to be able to manage the data accordingly and provide access whenever required.

To start the network traffic classification, data is first collected. This is the first step of the process. Data is captured in real-time and stored in a usable format. We used Wireshark as the data collection tool. While using the internet to surf, data flows were captured for about an hour and over 100,000 entries are produced.

Using Wireshark, the data is saved in a comma-separated values (CSV) format for further analysis. Feature extraction and feature selection are performed. The features include the length of the packet, source IP address, destination IP address, and the protocol. Since our primary target is to show how to develop an agent as a part of MANA-NMS system, we do not discuss on improving the design an ML. After feature extraction, the data is then sampled and divided into two sets, namely the training set and the test set with a data-split ratio of 80/20. During sampling, care should be taken to ensure that all the data samples have correct labels. In this case, the label is the protocol associated with each data packet.

After completing the agent design and using agents as a building block, the MANA-NMS model is designed as shown in Fig. 3.39. Incoming network traffic is collected and classified accordingly into three categories/classes. In our implementation, we considered high priority (HP), representing Ultra-Reliable-Law-Latency

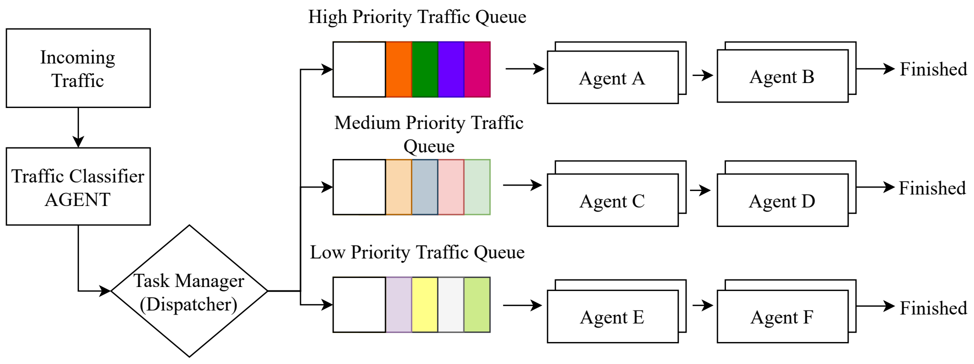

High Priority Traffic Queue
Incoming Traffic
Traffic Classifier AGENT
Task Manager (Dispatcher)
Medium Priority Traffic Queue
Low Priority Traffic Queue
Agent A
Agent B
Agent C
Agent D
Agent E
Agent F
Finished
Finished
Finished

FIGURE 3.39: Proposed MAS Model.

Communication (URLLC) services, medium priority (MP) representing Massive Machine Type Communication (mMTC) services, and low priority (LP) representing Extreme Mobile Broadband (xMBB) services. Once classified, the traffic is passed to a task manager (dispatcher), what decides which server or agent should handle the traffic based on the scheduling policy. We used a first-in-first-out (FIFO) policy when the traffic belongs to the same class. Service traffic is then scheduled on the corresponding service agent for service processing.

Task Manager and Queuing

Once the traffic is classified into the different classes, it is sent to the task manager. The task manager works as a supporting service agent in the MANA-NMS. In the task manager, each class of traffic is termed as task. The role of the task manager is to decide which server/agent is going to process the service. This decision can only be made if there exists a policy.

Scheduling Policy

The policy employed here is the "Join Class Related Queue" policy. This is a dynamic policy that sends jobs/tasks to the queue related to a server/agent which subscribes to that job (traffic class). For example, once a task belonging to the high priority traffic class is released by the task manager, this task joins the queue of the server/agent that subscribes to handle the high priority traffic class.

Queuing Model Employed

Each queue is modeled as an M/M/1 type of queue where the M stands for Markovian. The queuing model used is FIFO implying that the first job to join the queue is processed first and the rest follow in order of earliest arrival. After the task manager releases a task/job, the job is sent to one of three queues depending on the traffic class it belongs to. There are three queues and therefore, three classes of agents e.g. HP, MP, and LP.

Evaluation Results

The results reported in this section include a quantitative analysis of performance measurements of all NTCA designs, namely SVM, Decision Tree, K-NN, and Naive Bayes. The performance metrics include classification accuracy, training latency, and

TABLE 3.6: Classification Accuracy of the different Network Traffic Classifier Agent Designs.

3*k_{th} run of algorithm	Classifier Agent Designs			
	K-NN (%)	Decision Tree (%)	SVM (%)	Naive Bayes (%)
1	98.95	99.6	94.77	88.01
2	98.87	99.44	94.45	88.01
3	98.79	99.52	94.53	89.14
4	98.79	99.6	94.53	87.61
5	98.63	99.44	95.17	88.42
6	98.63	99.52	95.09	88.09
7	98.63	99.36	94.21	87.77
8	98.87	99.68	94.85	88.9
9	98.87	99.44	94.93	87.93
10	98.95	99.44	94.93	88.74
Mean	98.798	99.504	94.746	88.262
Variance	0.01628	0.00967	0.09394	0.26282

classification latency. The NTCA designs are compared and evaluated according to these performance metrics.

Classification Accuracy

The NTCA designs presented are analyzed using a 80/20 split of the data set into training and test sets respectively. The classification accuracy is affected by a number of factors and these include the number of features considered, the data set size and correctness of features considered to obtain the target class.

The NTCA designs are evaluated by running their respective algorithms 10 times. The mean and variance of the accuracy values was calculated and reported in Table 3.6. It demonstrates that the Decision Tree NTCA design is the most robust of all the classifier agent designs since it has the smallest variance and the highest mean accuracy when the algorithm is performed numerous times.

Training Latency

Training Latency is the time taken to train the traffic classifier model. Training latency is a measure related to the real time performance of the NTCA design considered. Less latency means better agent design performance.

The mean training latency values for the Decision Tree, K-NN, Naive Bayes, and SVM NTCA designs are 154.3, 154.9, 155.1 and 162.2 *ms* respectively. It should be noted that the Decision Tree NTCA design has the least mean training latency of 154.3 *ms*. Fig. 3.40 shows the training latency of all the NTCA designs over a number of training trials. It is evident that Naive Bayes, Decision Tree, and K-NN NTCA designs take a relatively shorter time to train a model as compared to the SVM NTCA design.

Classification Latency

The time it takes to categorize a single instance of previously unknown data is known as classification latency. Because latency is linked to a system's real-time

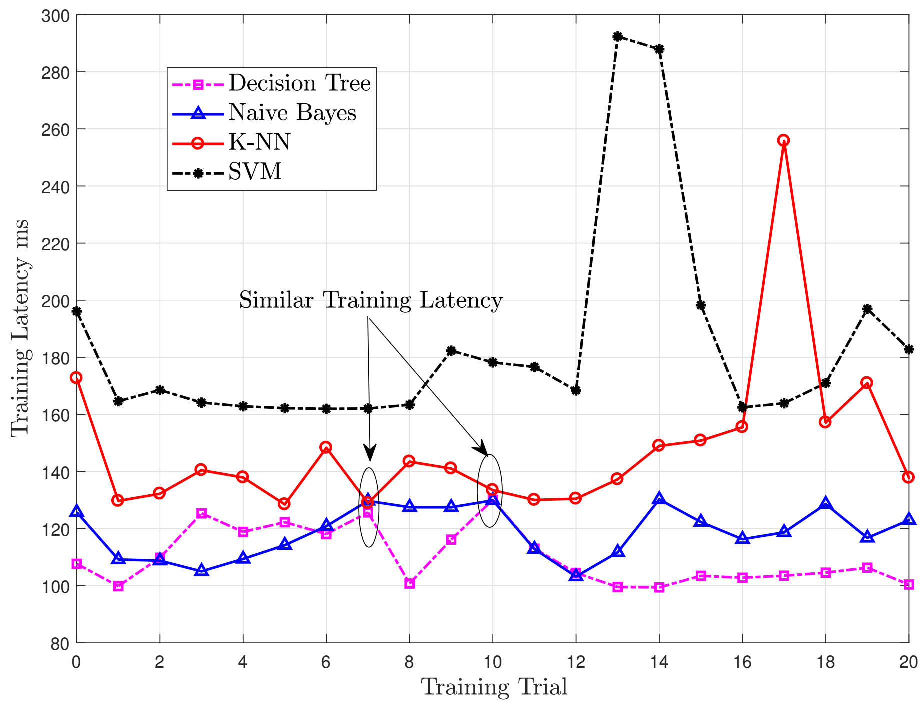


FIGURE 3.40: Comparison of the Training Latency of the Network Traffic Classifier Agent Designs.

performance, it is an important performance metric for NTCA designs. Less latency is better since the agent's input-output transformation takes the shortest time possible. A comparison of the classification latency of the NTCA designs observed over a simulation period of one hour was evaluated and presented in Fig. 3.41.

On average, the K-NN, SVM, Naive Bayes, and Decision Tree NTCA designs have classification latency values of 28.01, 3.668, 2.773, and 1.593 μs. Generally, SVM, Naive Bayes, and Decision Tree NTCA designs have relatively small mean classification accuracy of less than 4 μs. However, the Decision Tree NTCA design has the lowest mean classification latency of 1.593 μs.

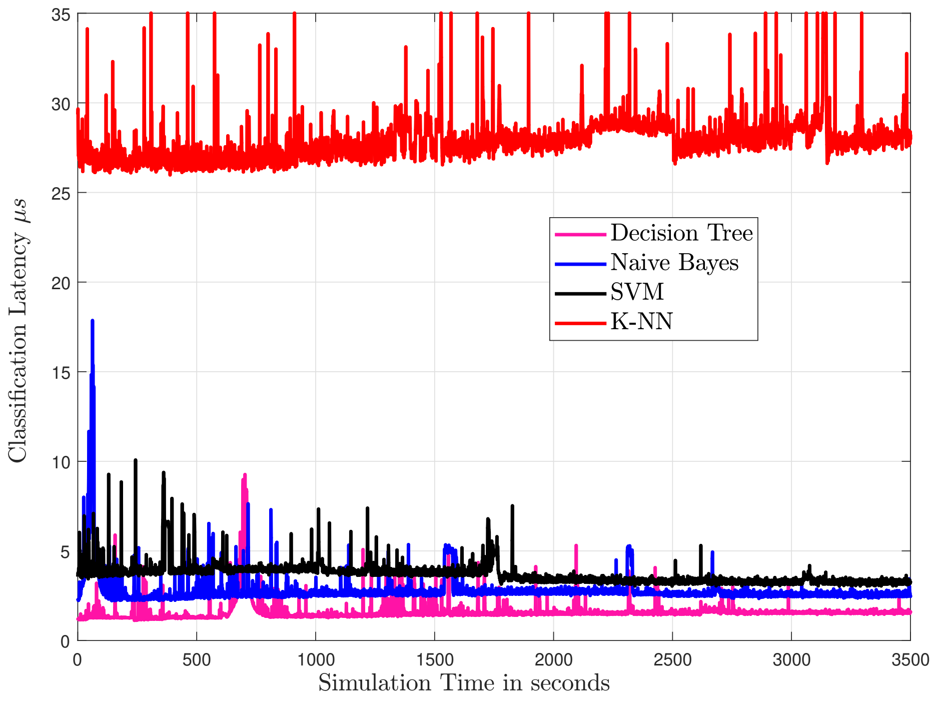


FIGURE 3.41: Comparison of the Classification Latency of the Network Traffic Classifier Agent Designs.

Chapter 4

SDN Controller Decomposition Using Microservice

4.1 Introduction

Note:this is a collaboration work between the University of Trento, University of Bologna and Technical University of Dresden. The testbed is developed at the University of Bologna. My contribution is the conceptual development, proposed architecture, testbed development guidance, and final article development.

As defined in the early section SDN is a networking paradigm that aims to give a definitive solution to break the limitations of traditional network infrastructure [236]. It breaks the vertical integration by separating the network control logic (i.e., the control plane) from the underlying routers and switches that forward the traffic (i.e., the data plane). With the separation of control and data planes, network switches have become simple forwarding devices, while the control logic is logically centralized in a controller entity, thus simplifying policy definition and network (re)configuration and evolution [135].

In particular, the SDN architecture consists of three layers [97]: data plane, control plane, and the application plane as an additional layer sitting atop them. Moving from the uppermost to the lowest layer, the *application plane* contains software applications to provide network services and performs ranges of functionalities such as Quality of Service (QoS), advanced security, and advanced routing. The *control plane* is the central agent which interfaces the application and data plane to implement applications network requirements: it communicates through the northbound interface to the applications and via the southbound interface to the forwarding devices. The *data plane*, is responsible for handling and forwarding packets and contains a group of data plane resources that can forward and manipulate packets. These resources include forwarding devices that have physical/logical interfaces to receive the incoming packets and forward them to an outgoing interface(s). The controller communicates with forwarding devices using several network communication protocols, in most of the cases the OpenFlow protocol [173].

The main issues of the centralized control plane range from latency constraints to fault tolerance and load balancing, to tackle those challenges, the distribution of the SDN controllers has been proposed to reduce typical issues of centralized controllers [22]. However, existing controllers are implemented as monolithic entities, even in the case of distributed deployments. In particular, in the case of distributed SDN controllers, there are replicas of the SDN controller, which means all SDN sub functionalities are replicated even if not all are necessary. For instance, Ryu SDN Controller [230], an open-source SDN controller implementation, provides a single piece of code installable on heterogeneous operating systems that enables the machine (or virtual machine) to act as an SDN controller. At the current time, all opensource and proprietary releases of SDN implementations adopt a monolithic software approach, which include ONOS [204], OpenDayLight [206], and Floodlight [93]. The main issue of monolithic implementations is that it does not allow network administrators and developers to choose SDN components and/or functionalities to (de-)activate for having the SDN controller functionalities according to SDN deployment and application needs in different scenarios. This results in limited flexibility in the network and creates multiple problems in terms of scalability, fault isolation, and latency. In particular, future 5G network infrastructures will leverage the network softwarization and network slicing concepts using SDN and Network Function Virtualization (NFV) in 5G [23]. However, in some scenarios system constraints can be very strict, such as in Industrial 4.0 and 5G Tactile Internet, which require a high rate of reliability and low latency communications [168] [2], and therefore, the adoption of a monolithic SDN deployment may result not suitable.

At the same time, the legacy definition of the SDN reference architecture does not mandate the internal composition, implementation, and design of an SDN controller [97]. Thus, the SDN controller can be decomposed and implemented as a set of software components, running in a distributed manner. Specifically, it is possible to design the SDN controller as a composition of logical sub-functions, sharing the network service load and creating a robust system against failures. These sub-functions are loosely coupled units that can be executed in different and distributed computing platforms [3]. The possibility of decomposing the monolithic SDN controller and designing the controller as loosely coupled provides a possibility of flexible controller deployment.

Accordingly, some research efforts have been started to decompose an SDN controller into microservices. For instance, the ONOS project proposes µONOS, which is the next-generation architecture for the Open Network Operating System controller [96]. µONOS adopts a microservices-based architecture disaggregating the controller and the core itself as an assembly of various subsystems. However, µONOS has been specialized mainly for cloud datacenter scenarios by employing a service orchestrator, Kubernetes, to manage microservices that are realized as Docker containers.

Even if µONOS, faced the issues discussed above, the implementation is in an early stage that needs more works to provide a playground framework. In addition, their approach has some limitations: first, is limited to certain technologies, not all 5G compliant, for instance, Kubernetes instead of ETSI MANO or containers instead of VNFs. Second, inter-functionalities communication is limited to Google Remote Procedure Call (RPC), which does not give a fair degree of flexibility in certain scenarios. Finally, the implementation is not completed yet and that hinders the possibility to thoroughly test it.

To overcome all those limitations, we propose a novel microservices-based SDN controller decomposed architecture based on Ryu SDN Controller called MSN that has been specifically designed for next-generation 5G RAN Edge deployments and shows several original elements. First, it shows original design guidelines for implementing a microservices-based SDN controller; second, presents a novel decomposition architecture for SDN controller by showing the use of REST-API or gRPC or WebSocket as different possible interfaces between the decomposed and virtualized/containerized functions of the controller; third, it presents an implementation proposal using Ryu SDN Framework that is completely agnostic to particular technologies and is 5G compliant (e.g., ETSI MANO and Virtual Machines or Docker Containers); fourth, it presents an evaluation of the proposed implementation, which indicates the robustness of the system and the low latency achieved by showing a comparison of communication interfaces such as REST-API, gRPC, and WebSocket; finally, an open-source version of our proposed framework is available for the community at the link: `https://gitlab.com/dscotece/ryu_sdn_decomposition/`.

4.2 Background and Motivation

In this section, we briefly review the existing standard architectures for softwarized networks such as SDN and NFV by analyzing the large numbers of synergies between them. Once clarified the standard monolithic SDN controller, we present the motivational factors for decomposing the SDN controller. Finally, we provide a list of SDN functionalities and we motivate the benefits for distributing these functionalities as microservices.

Background

SDN and NFV are network softwarization paradigms that are transforming the network management and design approaches. Network softwarization is the mapping of hardware-based network functions into software. Network softwarization enhances the possibilities of innovation due to flexibility, programmability, virtualization, and slicing. SDN and NFV enable the traditional static network to be flexible paving the way for network innovation.

What we have discussed above implies the physical separation of the network control plane from the forwarding data plane. NFV is an architecture proposed by ETSI for network function softwarization. In other words, it is a softwarized implementation of network functions. The functions are traditionally implemented in preparatory hardware [189] such as firewall, load-balanced, deep packet inspection (DPI), and network address translator (NAT).

An attempt to unify and find a single architecture considering the two paradigms is done in [148]. The SDN controller provides the possibility of programming the network to have a virtualized network that NFV could use to orchestrate virtual functions that are deployed in a data center or distributed environment. Whereas, NFV could provide a virtualized SDN controller that can be deployed in a cloud. Such possibility provides flexibility and full network function softwarization. Figure 4.1 shows the unified SDN and NFV architecture.

4.3 Overview of SDN Controller Components

Existing legacy SDN controllers are typically implemented as the composition of various function modules and libraries in a single monolithic system [204, 206, 93, 230]. For instance, Ryu controllers have internal components such as event distributor, topology discovery, and firewall and libraries such as Netconf, NetFlow, and sFlow. The Open Network Foundation (ONF) defined the basic elements and conceptual framework for an SDN controller design [97]. As defined by ONF, the internal components of the SDN controller typically contain the following basic network elements and basic functions. The very basic network elements that an SDN controller has to manage are:

- Devices are units such as switches, routers, ports, and other physical networking units.

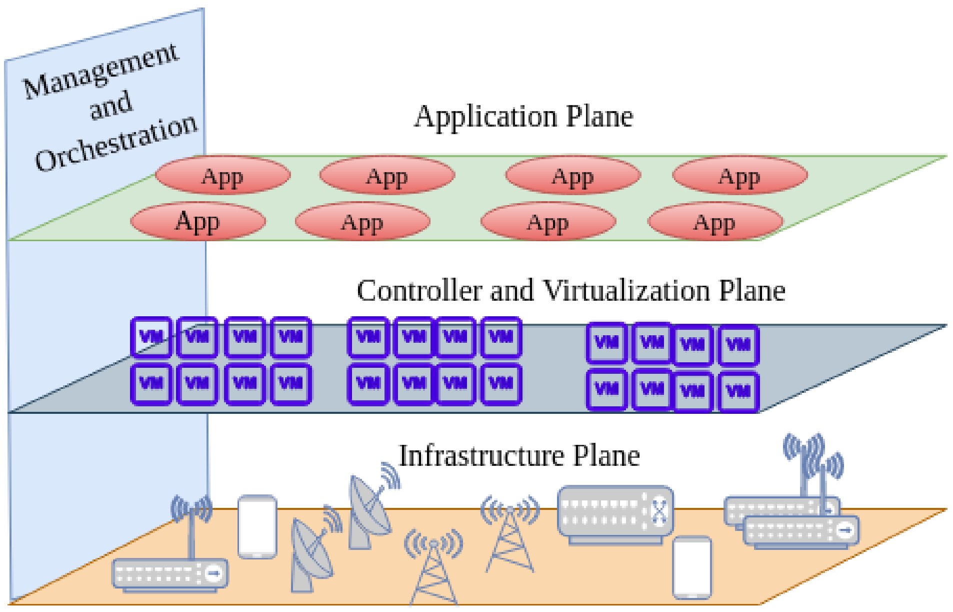

Management and Orchestration
Application Plane
App
Controller and Virtualization Plane
VM
Infrastructure Plane

FIGURE 4.1: A Unified Architecture for SDN and NFV

- Links are physical or wireless interconnections that connects one physical/logical port to another physical/logical port(s).
- Hosts represent the end devices such as computers.
- Packets/flows are the fundamental units of information in user and management services, at the network layer.

An inventory of these elements is registered and their state is updated in the controller's database. The common functions are:

- Topology management is managing a topology and determining which nodes and edges are present in the topology.
- Device and link discovery and management is a mechanism to configure and incorporate new devices into the network system.
- Route management is determining the path for a packet/flow to route through the network from the source to the destination. It computes the path for a given packet of flow based on the packet information.
- Routing/forwarding rule-setting enables the packet to route from the source to the destination based on the computed path.
- Performance monitoring is the mechanism of ensuring the performance of a network such as QoS for a given service.
- Network-state management is the management of the network information such as links status, available path, the available device along with their status, etc.

Depending on SDN controller types, the internal components of a controller may vary. Figure 4.2 depicts the ONF's SDN architecture. The functionalities mentioned above are confined in the Control Plane as part of the SDN controller. In addition, there are possible to have a set of sub-functions that we can consider as

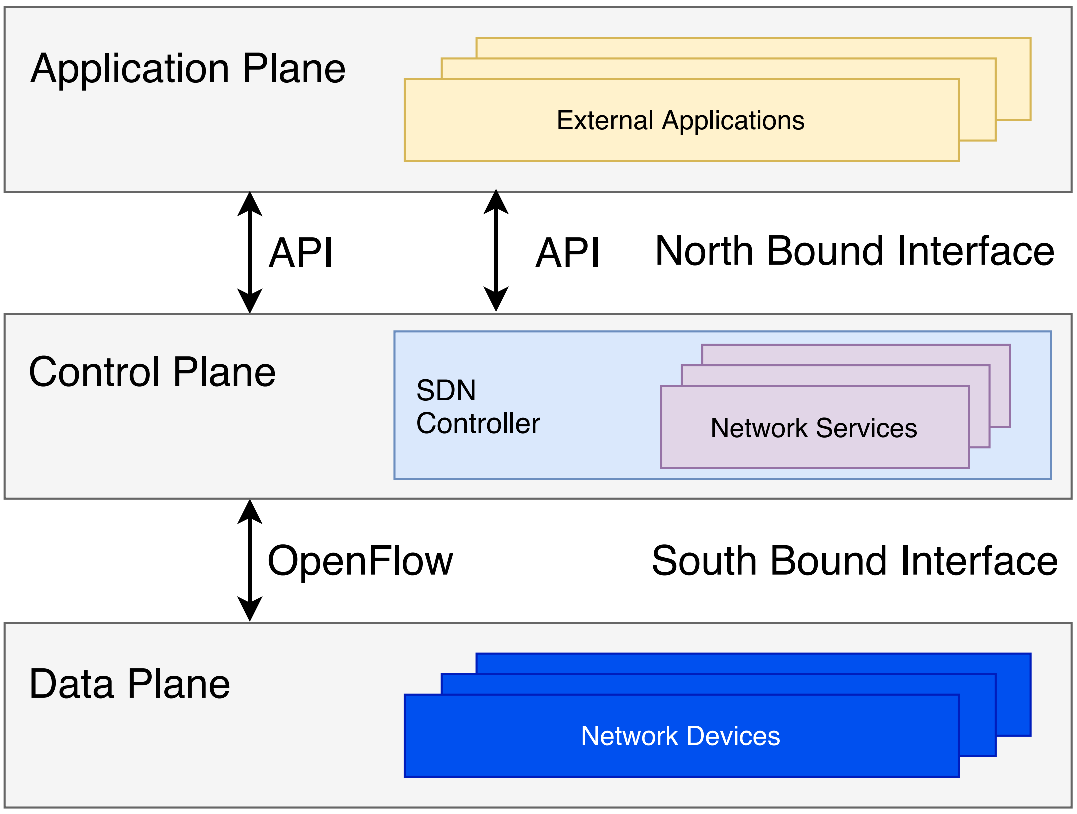


FIGURE 4.2: ONF's Software-Defined Network Architecture

additional/high-level functions. These functions in most existing SDN controller implementations are implemented as external applications confined in the Application Plane. In particular, the SDN controller supports a set of APIs (via the North Bound Interface) that make it possible to implement external network services. The most common SDN-based external applications are:

- Virtualization and slicing;
- Tenant creation and tunneling;
- Traffic flow measurement and statistics (telemetry);
- Performance monitoring;
- Firewall and security;
- Network address translation;
- Load balancing.

4.3.1 Ryu SDN Framework

Due to its simplicity and components-based architecture, the Ryu SDN framework is the starting point for our MSN implementation. In particular, a Ryu application consists of a Python script that extends the RyuApp base class and implements the Observable and Observer interface. These interfaces allow the application to interact with the event-based communications in the Ryu framework. For instance,

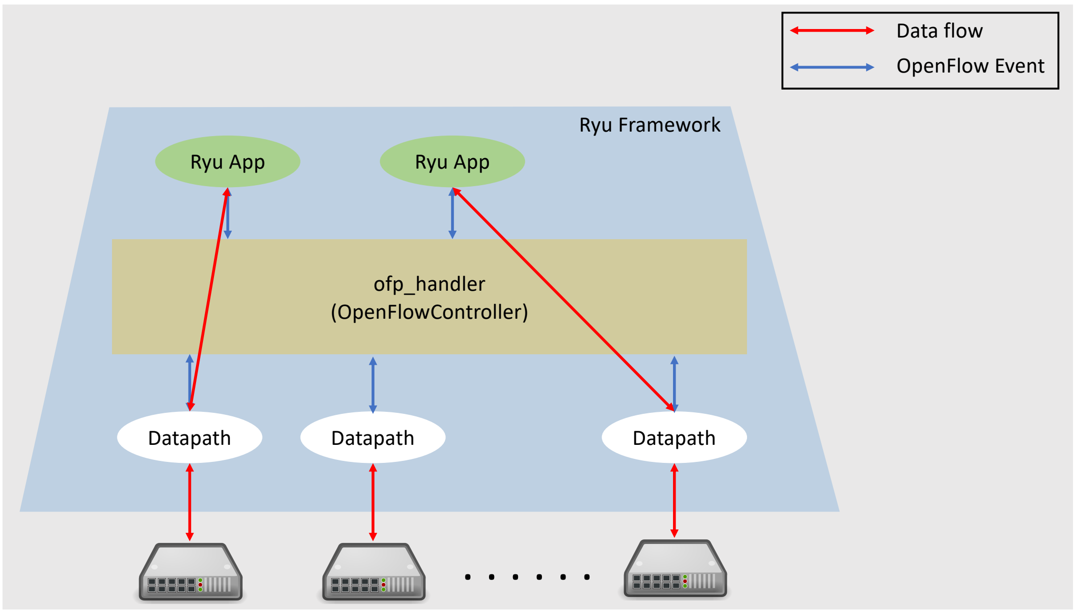


FIGURE 4.3: Ryu Internal Monolithic Logical Architecture

the *OFPPacketIn* event is the event generated when the switch sends the packet to the controller. This event invokes the subscribed functionality that can process the packet and can create the OpenFlow rule. First, the Ryu Framework starts the Application Manager that loads all the applications and registers the associated events. The most important application in the Ryu Framework is *ofp_handler* which allows the framework to interact with OpenFlow protocol (*OpenFlowController* class). To efficiently communicate with switches, Ryu Framework creates a virtual representation of switches called *Datapath*.

Figure 4.3 shows the logical architecture of the monolithic Ryu SDN Framework. The *ofp_handler* operates as the event dispatcher of the Framework and manages the Datapaths. In particular, it manages the *Hello* and *Echo* messages and updates the status of the ports when necessary. The Ryu apps, once invoked by an event can reply directly to the Datapaths. By default, if nothing is specified, Ryu starts with only *ofp_handler* as application. The *ofp_handler* is the core-fundamental component of the Ryu SDN framework as it contains all the basic SDN functionalities.

In recent years, there has been a lot of interest in SDN-based mobile networks, and several papers are proposing SDN-based mobile network architectures and listing the benefits they can bring to the mobile industry [108]-[200]. In particular, the high numbers of researches and the high interest in this topic have led to an evolution of the traditional SDN architecture and due to the widespread of the IoT, the SDN paradigm has started to be used to manage the IoT in several domains including smart city [7], smart home [258], smart health [164] and so on. The SDN paradigm helps the IoT networks to challenge several issues such as latency, reliability, privacy, flow control, etc. However, it still has some issues caused by the logical centralization of the SDN controller, the main of them being scalability and robustness [199].

There are also numerous SDN controller implementation both from the open-source community and from commercial vendors. Typical examples are NOX, POX, Ryu, OpenDayLight, and ONOS. Even if the core principle of all SDN controllers is the same, each of them has a slightly different implementation approach. NOX is the first OpenFlow-based SDN controller written in C++[197]. In the early exploration of the OpenFlow and SDN space, NOX has been the basis for many research and development projects. The NOX internal components mainly contain event handlers to receive and dispatch events such as incoming packets. POX is similar to NOX with a Python-only implementation. It is considered as a general, open-source OpenFlow controller [215]. Ryu is also a component-based SDN controller[230]. OpenDayLight is a collaborative open-source controller[206]. It is a modular, extensible, scalable, and multi-protocol SDN controller deployment. ONOS is also an open-source SDN controller [204]. The ONOS SDN controller software is written using Java and it provides a distributed SDN application platform atop Apache Karaf OSGi container. The controller has been designed to operate as a cluster of nodes that are similar in terms of their software stack and can endure the failure of individual nodes without causing disruptions in its ability to control the network operation.

All of above SDN implementations are based on a modular approach. However, all of them are based on monolithic architecture. NFV has changed the way we deploy network functions [189]. It enables easier, flexible, and dynamic deployment of a given network functions. SDN and NFV are complementary technologies [189][97], but SDN controller function could be considered as a network function and it can be deployed as a VNF in a container [189]. However, the overall controller is a cumbersome and monolithic process. ONF specification indicated a possibility of implementing the SDN controller as either monolithic or decomposed in microservices [97]; however, only a few recent efforts have started exploiting this second possible design choice.

In wireless cellular networks, a recent article [41] showed how to split BaseBand units (BBU) of wireless RAN deployment. The authors propose to split it into different configurations; in each configuration, the functional units of BBU are split to be deployed as a virtualized functions. The BBU is virtually stored in a network cloud and accessible, as a shared resource. Similar to other network functions, such as BBU in wireless networks, SDN controller could be decomposed [61].

The first work showing an externalization of packet processing in SDN is presented in [61]. As an extension of this work, the author in [62] provided steps that are required to migrate from a monolithic to a microservice-based architecture. The functional components are distributed as microservices and a gRPC is used to communicate between the core modules and external components or applications.

µONOS is the latest solution proposed towards a standard architecture for distributed and split control plane. The µONOS project aims at creating a new generation of SDN architectures based on ONOS, splitting it into a set of microservices. These split functionalities are deployed as Docker containers and managed by Kubernetes orchestrator. The µONOS project is relatively young (it started in October 2019) and it is based on the P4/P4runtime [209] protocol which is a different control protocol compared to the standard de-facto for the SDN paradigm, namely, the OpenFlow protocol. P4 protocol can emulate the behavior of OpenFlow; in addition, the communication between functionalities is via gRPC-based protocols, including gRPC Network Management Interface (gNMI) for network management interface configuration and gRPC Network Operations Interface (gNOI) for network command operations. However, the µONOS implementation is still in its infancy and there is still no available implementation to play with. Moreover, and most important, some implementation choices are not compliant with ongoing 5G standards. First, µONOS has limited integration with and support for ETSI-NFV standards because it neither implements the OpenFlow abstraction nor that of legacy network elements. In particular, the use of Kubernetes does not allow an easy integration within the 5G Edge architecture which requires ETSI-based protocols such as ETSI-MEC and ETSI-MANO. On the contrary, MSN framework exploits ETSI-NFV standard to define its architecture and ETSI-MANO for managing microservices, which allows MSN to easily operate within 5G-based networks. Second, in µONOS the communication between microservices is based on gRPC, a Remote Procedure Call framework developed by Google, which allows network entities to communicate (once defined an agreement) by serializing data with the Protocol Buffers, another Google solution. This results in a not so easy interaction with most available third-party applications which are using more open protocols such as REST-APIs. Because of that, in our MSN solution, we defined a generic communication module between microservices that allow users to choose the network communication technologies according to scenario requirements. This operation can be done at developing time, but, thanks to the dynamicity of containers/VMs orchestration via ETSI-MANO, it is possible to change dynamically the network communication technology. Finally, µONOS is based on the ONOS SDN system that forces users to have previous knowledge on it. Also, this may result inefficient in several scenarios where network entities are not powerful enough to run the (rather heavy) ONOS system such as the Industrial IoT scenario. To overcome this, MSN framework provides general guidelines to decompose an SDN system that is completely agnostic to specific SDN software and communication means.

In general, to the best of the authors' knowledge, the MSN solution is the first seminal work implementing complete guidelines for the SDN controller decomposition in microservices that fit 5G requirements including an implementation based on VNFs, Docker containers, and ETSI-MANO.

4.4 Motivation for SDN Controller Decomposition

Recently, the application of distributed SDN controllers has been widely studied in the literature, from different deployment perspectives. The main efforts were on the applications IoT device for smart city, disaster management [35, 301, 184], and Industrial Internet of Things (IIoT). In the following of this paper, we focus on the application of distributed deployment for IIoT. This because is a challenge scenario due to very strict requirements in terms of latency, reliability, and scalability. In particular, IIoT is a new paradigm in Industry 4.0 and it consists of the remote operation of machines, computers, and robots enabling intelligent industrial operations. Moreover, it is aimed at complete automation of the manufacturing process, from the raw material input to manufacturing, storage, distribution, and end-user marketing. In such scenarios, various heterogeneous devices and users are involved [200]. IIoT network requires real-time controlling, e.g., to control robots. These applications require, resilient, dynamic, and autonomous networking with low-cycle times (around 100 ms), and a high-reliability rate (close to 99,99%) [168, 2]. These requirements are very difficult to achieve with the existing SDN centralized controller deployment, especially for a large network. It is because SDN deployment incurs in propagation latencies as they have to be deployed at the center, which could be at a significant distance from the network device or forwarding switches to be managed. Therefore, despite the expected benefits of centralized controller design, it raises many challenges, including scalability and reliability.

The existing technique proposed to alleviate this problem is the hierarchical deployment of the SDN controller. Using the physical decentralization of the control plane approach, it is expected to address the scalability and latency problems [200].

However, such physically distributed, but logically centralized, systems bring an additional set of challenges. First, the distributed deployment of a monolithic SDN controller requires (unnecessary) replication of code so to deploy the whole SDN controller at each location. That means whenever a new controller is deployed the whole SDN system has to be replicated in the distributed location. In other words, the monolithic system further provides a granular level challenge in terms of increasing the required functionalities. As the SDN system's internal modules are tightly coupled, it is difficult to dynamically increase the serving capability of the controller without adding a new SDN controller. In other words, the whole SDN has to be replicated in response to the workload demand that could have been performed by increasing just the required functional modules of the SDN controller. Furthermore, in terms of resilience, the monolithic system also has further disadvantages as a single controller fails in a given local area, it has to use the central controller which is further away from the center creating latency and congestion [283]. This is because instead of instantiating the specific function that leads to failure, either it has to instantiate the whole SDN controller or contact the nearby controller. See Table 4.1 for comparison between monolithic systems and microservice system.

However, if we can decompose the monolithic SDN controller into sub-functions, we could deploy only the required functionality in the required location as per network size and network management workload demands. Moreover, dynamic response to dynamic network demand is possible by dynamically instantiating the required functions of the controller components to meet the service and network demands. This means dynamically scaling the controller capability with the dynamic service demand.

Therefore, a decomposed SDN controller deployment could provide a flexible and efficient local deployment of required controller's functionalities, while deceptively scaling controller's unctions on demand. This would potentially be advantageous in terms of latency and reliability, due to reduced code size and the flexible scaling of resources and controller functions. It would become possible to dynamically increase the number of functions and resources, which could be a horizontal and vertical extension of functions deployed as virtual network functions (VNFs) to meet the service demands. A typical application of a decomposed SDN scenario is depicted in Fig. 4.4. The SDN controller with the minimum required functionality could be hosted in an edge data center.

4.5 Microservices-Based SDN Controller Decomposition

The monolithic SDN controller is easier to develop and deploy. However, since a monolithic SDN controller is built as a single and indivisible unit, updates or changes are very difficult because they require the replacement of the whole stacks of the control system. Moreover, handling a huge codebase, adopting new technology such as AI, dynamic scaling, deploying, and implementing changes is very difficult. This is a big disadvantage in the era of functions *containerization* and *cloudification* where features of loose-coupling, distributed deployment, and dynamic scaling of resources are required. In general, the monolithic SDN controller has disadvantages of scalability, reliability, and reusability.

In principle, monolithic SDN controllers have pros and cons compared to microservices. Microservices are a means of creating loosely-coupled sub-functions or sub-services, replacing a large software system. So far we have mentioned its disadvantages. An alternative approach that we have indicated so far is microservices. A

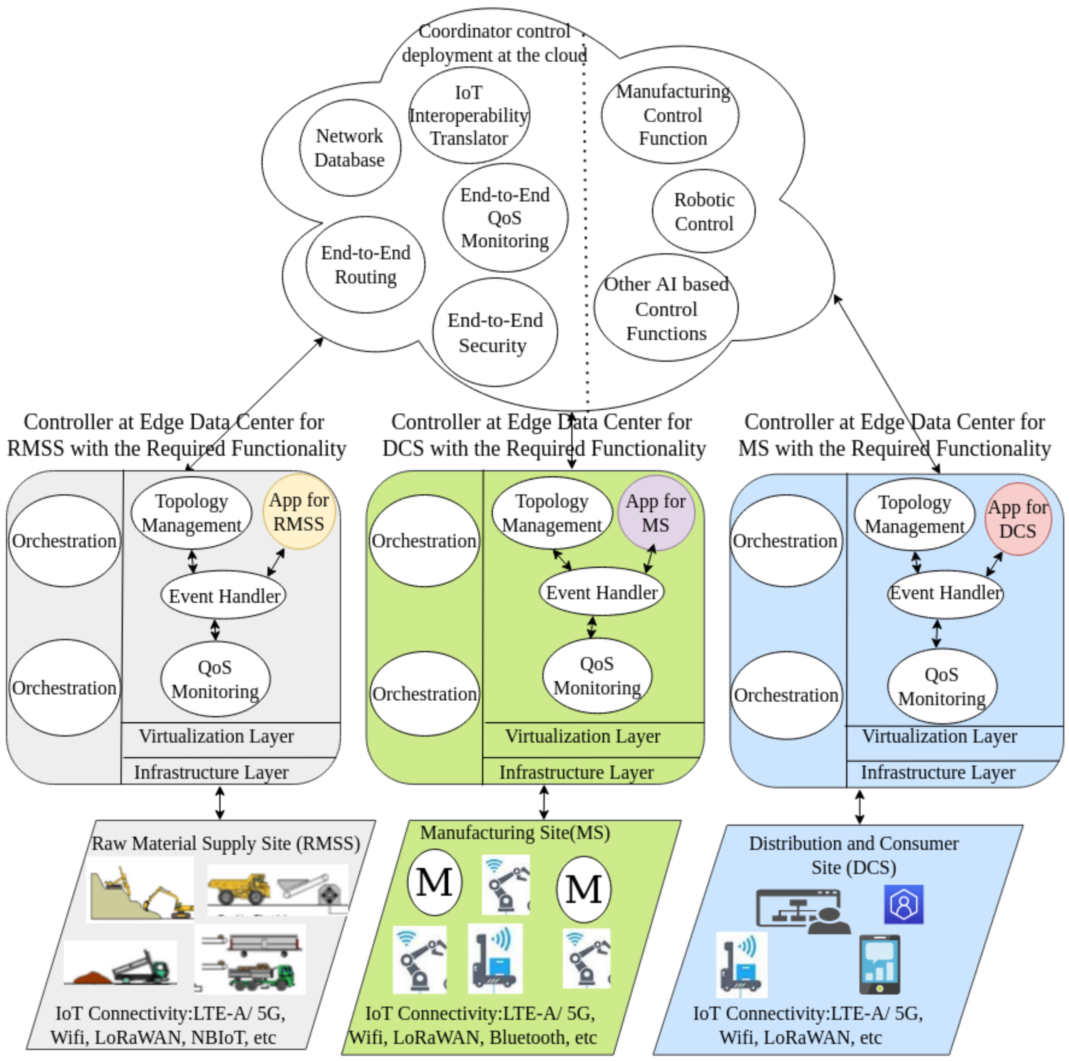

Coordinator control deployment at the cloud
Network Database
IoT Interoperability Translator
End-to-End QoS Monitoring
End-to-End Routing
End-to-End Security
Manufacturing Control Function
Robotic Control
Other AI based Control Functions
Controller at Edge Data Center for RMSS with the Required Functionality
Controller at Edge Data Center for DCS with the Required Functionality
Controller at Edge Data Center for MS with the Required Functionality
Orchestration
Topology Management
App for RMSS
Event Handler
QoS Monitoring
Virtualization Layer
Infrastructure Layer
App for MS
App for DCS
Raw Material Supply Site (RMSS)
IoT Connectivity:LTE-A/ 5G, Wifi, LoRaWAN, NBIoT, etc
Manufacturing Site(MS)
M
IoT Connectivity:LTE-A/ 5G, Wifi, LoRaWAN, Bluetooth, etc
Distribution and Consumer Site (DCS)
IoT Connectivity:LTE-A/ 5G, Wifi, LoRaWAN, etc

FIGURE 4.4: Microservices-based SDN deployment blueprint in IIoT Scenarios

TABLE 4.1: Comparison between SDN Monolithic architecture and Microservices-based SDN architecture [283]

Design Principles	SDN controller as monolithic system	SDN controller as Microservices
Scalability	Monolithic SDN controller is difficult to scale because it requires replicating overall SDN controller for smaller increase in demands. This is because tight coupling of internal structure of the Monolithic SDN controller. Moreover, it is very hard to upgrade as it requires working on the big system instead of replacing or maintaining just the failed functions.	Microservices-based SDN controller can be placed in a container or any virtual environment. Using function sequencing, dynamic instantiate of containerized services, and function orchestration, it is possible to recreate the SDN functionality, which can easily scale by adding only those components that require additional resources.
Cloud Readiness	Monolithic SDN controller is bulky to deploy the system in a containerized environment.	Microservices can easily be deployed and orchestrated in containerized environment. Therefore, a microservice-based design of SDN controller enables easy deployment in a distributed environment such as cloud
Loose Coupling	Monolithic SDN controller's internal modules are tightly coupled which prevents it to be deployed in a distributed environment without replicating the controller. Wherever and/or whenever SDN controller functions are needed, the whole system must be deployed instead of the required functions only	On the contrary to Monolithic based controller, SDN controllers based on microservices have loosely coupled components which enables flexible deployment in a distributed environment for dynamic scaling along with the dynamic service demand
Maintenance	If a Monolithic SDN controller internal component fail, locating the problem to make changes is difficult and may take a lot of time. This is because the SDN controller is a big complex set of code that are tightly coupled	Instead, in the case of microservice-based design of SDN controller has enough decoupling of the functional components to identify, isolate, and replace with new function of that particular function without requiring to replace or work on the whole system. I means, If one part of the system fails, mostly only a single component of the SDN controller function may damaged, and it can be isolated and replaced/fixed. This could be by instantiating a new container for example
Component reuse	For a monolithic based SDN controller, it is difficult to reuses the sub-components as every function or internal modules are part of a single system	However, microservice-based components of an SDN controller functions could be reused and orchestrated to be deployed with other functions for dynamic response to the workload increase. So microservice-based SDN controller functions reusable as components are loosely coupled, independently implemented and deployed

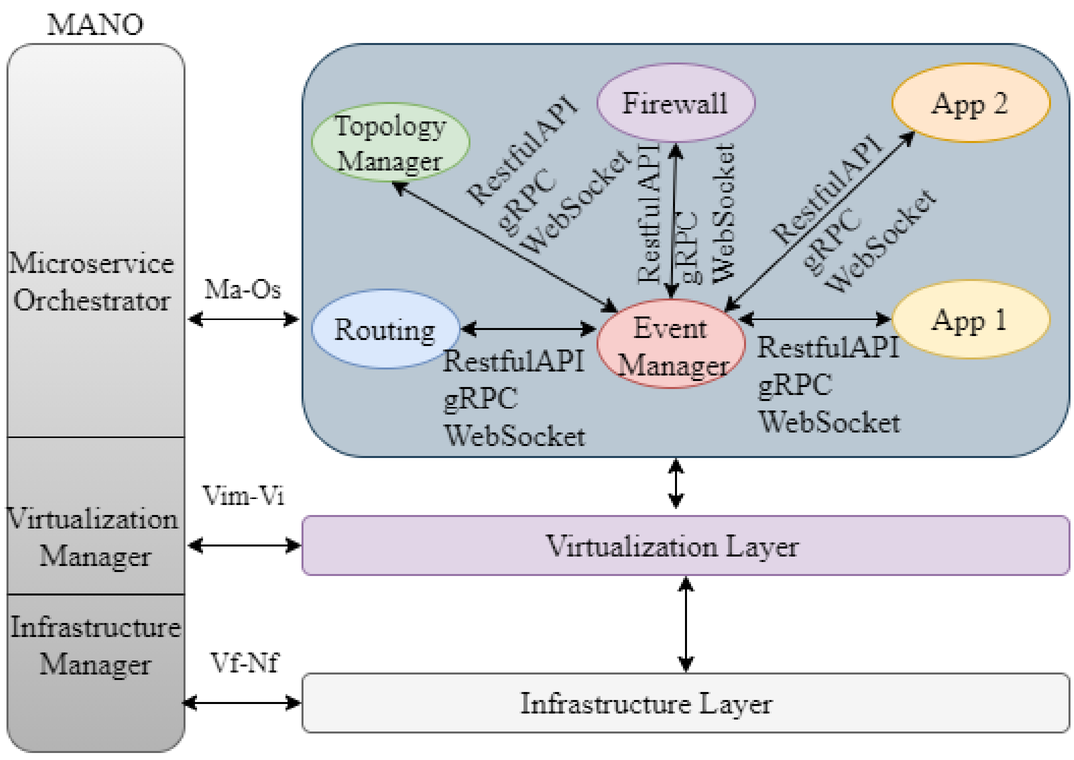


FIGURE 4.5: Architectural Overall of Proposed Microservices-based SDN Controller Decomposition

multi-agent system is also another alternative to have a decomposed management system. Since we are focusing on a microservice-based decomposition technique, we will limit our discussion to only microservices instead of multi-agent approaches, which is discussed in detail in [16].

4.5.1 Decomposing SDN Controller

This section provides the proposed decomposed architecture of an SDN controller, presenting a functional definition of components and interfaces. The proposed SDN controller decomposition architecture shows the decoupling of internal components of the SDN controller to be deployed as a microservice.

4.5.2 SDN Controller Internal Components as a Microservice

The main principle to retain in decomposing an SDN controller is that the network information and state should be synchronized and self-consistent providing a global view of the network. That allows an independent implementation and components reuse. We consider a decomposition of SDN controller, as depicted in Figure 4.5 which shows a decomposed three-layer SDN architecture reflected in an NFV architecture [189].

Figure 4.6 depicts our proposed deployment architecture of the decomposed SDN controller, based on microservice architecture. The main principle behind the architecture is the use of microservice based functions to replace the monolithic SDN controller and develop it as a lossless coupled composition of containerized services.As discussed in Section 4.3, we identified and defined the internal components of the SDN controller that could be developed independently as a microservice. This requires delineating the system based on specific service functions. In other words it should be possible to specifically define as function that could independent be developed as microservice and deployed in a virtual environment or container.

This also mean that the control layer core sub-functions are decomposed into sub-functions and implemented as microservice and deployed as VNFs in containers. Each sub-function is developed as a microservice and creates an independent and autonomous service unit. These microservices are able to independently perform the required service function. The independence of microservices would provide the possibility of deploying them in a distributed environment while scaling them as per the service workload. This may require instantiating multiple instances of the same service or new instances of additional microservices.

Once the necessary functions are developed as microservices, the SDN system requires a service aggregation to deliver a final functional system. Therefore, the final controller system becomes the organization of the independently developed microservices to create the equivalent SDN controller functions. As indicated above, the controller's components can be executed on arbitrary computing platforms on distributed and virtualized resources such as virtual machines (VMs) or containers in data centers. The loosely-composed system can be viewed as a black box, defined by its externally-observable behavior, emulating the original monolithic SDN controller. However, a distributed implementation must consider maintaining a synchronized and self-consistent view of network information and states. The independently developed microservices based functions could be orchestrated by a standard orchestrator, such as MANO. This would create a service function chain to equivalently perform the legacy SDN controller's functions.

As can be seen from Figure 4.5, the upper layer is a pool of independently implemented microservice based SDN components such as topology manager, event handler, and other applications. Each component performs a specific function such as traffic routing, topology management, and even handling. It is possible to categorize functions as basic SDN controller functions and additional functions or applications. Basic SDN controller functions are mandatory to provide the minimum possible function of the SDN controller.

4.5.3 External Applications as a Microservice

In addition to the network services, external applications could be incorporated to extend the controller basic functionality [97, 96]. Depending on the network to be controlled, various types of applications could be implemented in the application layer such as additional QoS service, traffic predictions, traffic classification, slicing, firewall, and novel deep packet inspection, see Figure 4.6. All these functions can be developed based on microservice and could be considered as VNF in container which can be deployed in a distributed environment. In doing so, we are effectively eliminating the traditional delineation between the control layer and the application layer. This is interesting concept to notice as the legacy architecture of the SDN controller has three layers, which are forwarding, control, and application, see Figure 4.2. However, in the proposed MSN framework there is no apparent difference between an SDN internal function and network application that are deployed as VNF in containers. This is because all components could be implemented as loosely coupled microservices, and running in a container that can be deployed anywhere. Our implementation is a testbed showing this by splitting the Ryu controller into two separate functions that are deployed in a docker container.

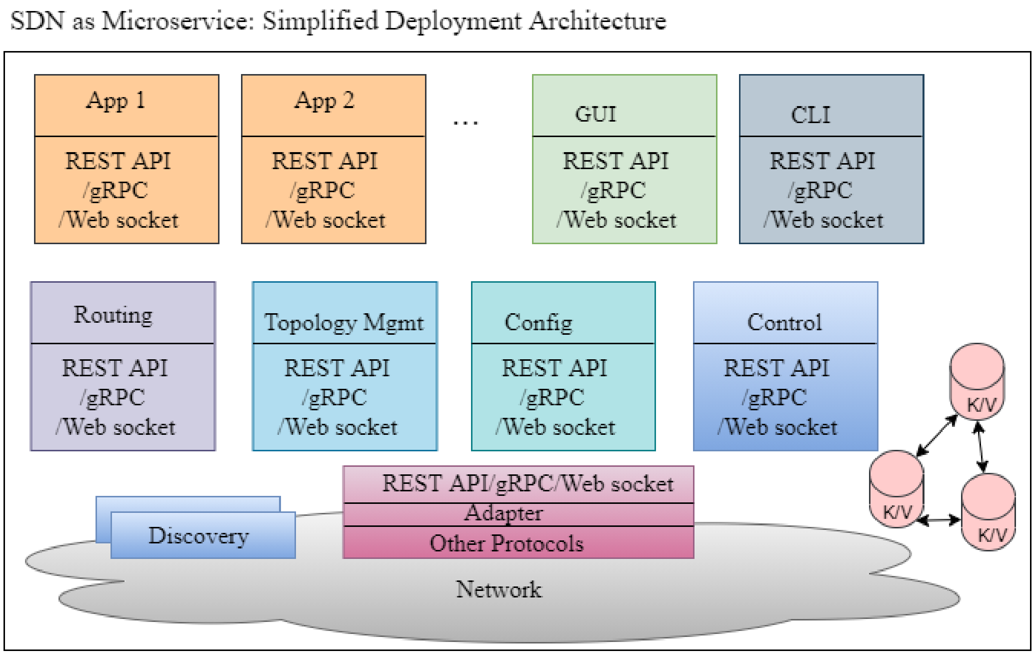

SDN as Microservice: Simplified Deployment Architecture
App 1
REST API /gRPC /Web socket
App 2
REST API /gRPC /Web socket
...
GUI
REST API /gRPC /Web socket
CLI
REST API /gRPC /Web socket
Routing
REST API /gRPC /Web socket
Topology Mgmt
REST API /gRPC /Web socket
Config
REST API /gRPC /Web socket
Control
REST API /gRPC /Web socket
K/V
K/V
K/V
REST API/gRPC/Web socket
Adapter
Other Protocols
Discovery
Network

FIGURE 4.6: Proposed microservices-based SDN controller sample deployment architecture

4.5.4 Communication Interface Between Decomposed Services

As indicated above, in our proposed MSN framework, the components can be deployed as VNF in a distributed environment as a web service. A web service is a service that can be called by an application. Therefore, the decomposed SDN controller sub-functions could be considered as web services, which can separate programs that are independent of other applications and can be run on different machines. Such functions communicate with each other or with the event distributor, such as sending and receiving event notifications through the communication interface. These communication interfaces between the decomposed and containerized applications are based on open-source communication interfaces. These generic communication interfaces are tested in our implementation which are REST, WebSocket, and gRPC. Each of them has its pros and cons in terms of latency for web-based services.

Communication Interface Between the decomposed Microservice based Controller Functions

Our proposed MSN framework uses RESTful API for as the communication interface. RESTful is an application program interface that uses HTTP requests to GET, PUT, POST, and DELETE data [138, 49]. It provides interoperability between different network application developers of the SDN controller sub-functions. These APIs can be used to facilitate efficient microservices-based function orchestration and automation of the network to align with the needs of different applications. RESTful API is a stateless architecture for data transfer. We chose REST for multiple reasons such as performance, scalability and, most important, is the standard declared in the 3GPP white-paper about Release 15 of 5G networks [1]. RESTful API also allows the support of large numbers of components and interactions among them which makes it ideal for IIoT deployment scenario indicated above. Moreover, RESTful API has a uniform interface which simplifies and decouples functions making it suitable for a microservice-based SDN function communication.

As a comparison with RESTful API while testifying our hypothesis on how the use of REST has advantages for SDN controller decomposition, we used gRPC and WebSocket [49]. gRPC is an open-source remote procedure call (RPC) system initially developed. It uses HTTP/2 for transport, Protocol Buffers as the interface description language, and provides features such as authentication, bidirectional streaming and flow control, blocking or non-blocking bindings, and cancellation and timeouts. gRPC is roughly seven times faster than REST when receiving data and roughly ten times faster than REST when sending data for this specific payload. This is mainly due to the tight packing of the Protocol Buffers and the use of HTTP/2 by gRPC. Moreover, WebSocket is another communications protocol that provides a full-duplex communication channel between the servers and the clients, using a single TCP connection. It was standardized by the IETF as RFC 6455 in 2011 [49]. It provides real-time communication between a client and the server.

Finally, we would like to indicate that the drawback of an SDN controller decomposition and deploying it as a distributed system could create an additional challenge of synchronization between components. Even if the decomposition has advantages compared to a centralized SDN architecture in terms of availability, resilience, and flexibility for a reconfigurable system, the distribution of functions imposes a continuous network state synchronization challenge. The network state database could be centralized or distributed. In other words, the network state is replicated or distributed between the controllers requiring repeated synchronization. In each case, maintaining synchronization is a challenge. However, the problem of synchronization of the database in a distributed system is a long-studied subject that could be considered for the case of decomposed SDN controller [70, 237]. For example, the existing controller synchronization strategies developed for distributed controllers improve joint controller decision making for inter-domain routing. Given existing solutions in the literature, in this work, we consider the proposed MSN system precisely synchronized.

This assumption is made reasonable by the system's characteristics. The various modules, composing the distributed SDN controller, are virtual containers placed in and running on servers and, more in general, on network computing hardware. The containers get the synchronization from the clocks of their hosting hardware. In fact, this network hardware is accurately synchronized via well-known standardized synchronization protocols like IEEE 1588 Precision Time Protocol (PTP) [163, 123], which has already been used to achieve a synchronization accuracy in the order of tens of nanoseconds.

4.6 Decomposed SDN Controller TestBed Implementation and Performance Results

For the evaluation of the MSN implementation, we proposed an implementation based on Ryu SDN controller [230] due to its component-based characteristics that blend well with the microservices-based SDN controller perspective. The following subsections introduce first the fundamentals of our proposed microservices-based decomposition framework and, finally, we present our experimental environment and the performance results of the MSN implementation.

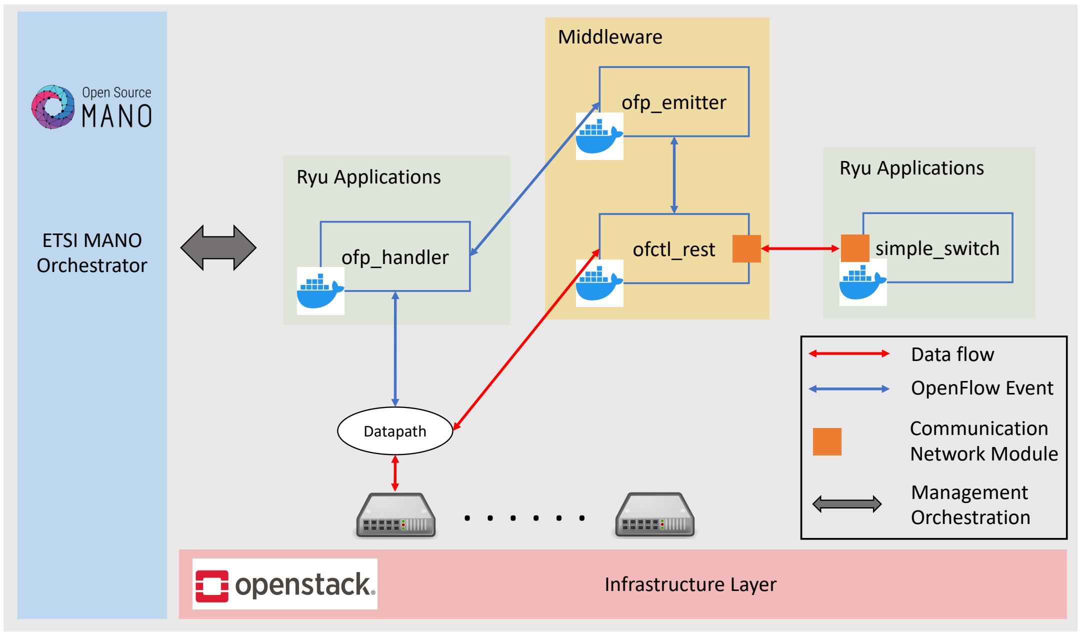


FIGURE 4.7: Ryu-based MSN Implementation Architecture

4.6.1 Decomposing Ryu SDN Controller

As theoretically described in the Section 4.5.1, we identified in the Ryu implementation the essential modules that describe a basic SDN system:

- Event Handler System Management: this module is in charge of catching an OpenFlow event and forwarding it to the destination. This module works reactively and may be considered as the core module for a decomposed SDN implementation.
- Routing System: this function is used to generate Flow rules to allow the network to exchange packets among nodes and switches.
- South-bound Management: this module allows the system to interact with the underlying system with several protocols.

The starting point for the microservices-based SDN decomposition is the characterization of the core part of the SDN system that allows the communication from network components to the applications (i.e.: from the control plane to the data plane). In particular, the externalization of that SDN core part allows the network to be observable and manageable from external processes. The proposed MSN implementation follows that principle and its implementation is provided to demonstrate the feasibility of using a middleware that allows the interaction between the core of the SDN and external processes.

First, we isolated the event emitter from the core of Ryu Framework and we created a support middleware module (the yellow block in the Figure 4.7), incorporating the REST APIs block with the emitter to be able to transform events in REST calls. The middleware is the fundamental block for a microservices-based SDN decomposition, precisely because connects the legacy SDN environment with external microservices. Second, we turned each Ryu App in a separate block (i.e., microservices) external to the Ryu Framework which can communicate with the Framework via REST APIs through the middleware. In this way, we transform an SDN functionality into an atomic block (microservice) releasing it from the whole SDN Framework. For implementation purpose, we leverage the already existing REST-based APIs in the Ryu framework, precisely the *ofctl_rest* module. Figure 4.7 shows the resulted Ryu-based MSN implementation architecture. The described approach can be used for different network technologies, not only REST-based, such as gRPC, WebSocket, RPC, and so on. What is changing is the block internal to the middleware (the *ofctl_rest* block in the Figure 4.7) module that connects to external microservices.

Once demonstrate the feasibility of the MSN framework, to improve reliability and scalability is important to leverage a solution like virtualization and/or containerization that results easy to orchestrate via an orchestrator. In our solution, we adopted Docker Container as containerization ecosystem, Open Source MANO for the orchestration and OpenStack as the infrastructure layer. We produced different Dockerfile for reproducing the architecture shown in Figure 4.7. In particular, we created a Docker container for our middleware that incorporates the *ofp_emitter* and *ofctl_rest* blocks inside and another Docker container for the event handler functionalities such as the *ofp_handler* block. Finally, Ryu Apps are considered as separated Docker Containers that include SDN functionalities including routing functionality or Firewall. For major details, we have all codes available at repo source: `https://gitlab.com/dscotece/ryu_sdn_decomposition/`.

4.6.2 Experimental Environment

Our performance evaluation aims to prove the feasibility of the MSN framework and evaluates its performance by proposing a benchmark of several communication technologies to enable needed interconnections and interoperability across microservices. The tests show results in terms of reliability, scalability, and latency of the system. To achieve this, we separated our testbed into two different parts: first, we tested the system latency introduced by splitting the SDN controller in microservices for different network interconnection technologies; finally, we tested our system to calculate the improvements in terms of reliability and scalability.

The performance of the MSN framework has been evaluated in a simulated testbed environment. We used the implementation details described in the Section 4.6.1. In addition, the OpenFlow protocol will be used for the forwarding plane of switches. However, it should be noted that the vision of the MSN framework is completely agnostic to any specific SDN implementation. Our testbed consists of a Linux workstation (Ubuntu Server 18.04 LTS) equipped with a 2x AMD Opteron(tm) Processor 6376 3.2GHz 16 cores processor and 32 GB 1600MHz DDR3 memory. We employ the following software, used to implement and test the proposed architecture:

- The Microstack version of the Openstack which plays the physical infrastructure of our testbed
- The docker community edition version 19.03.7 for dockerized microservices
- The open-source MANO release 8 as the system orchestrator
- Ryu SDN Framework
- Python 3
- Mininet for creating a virtual network

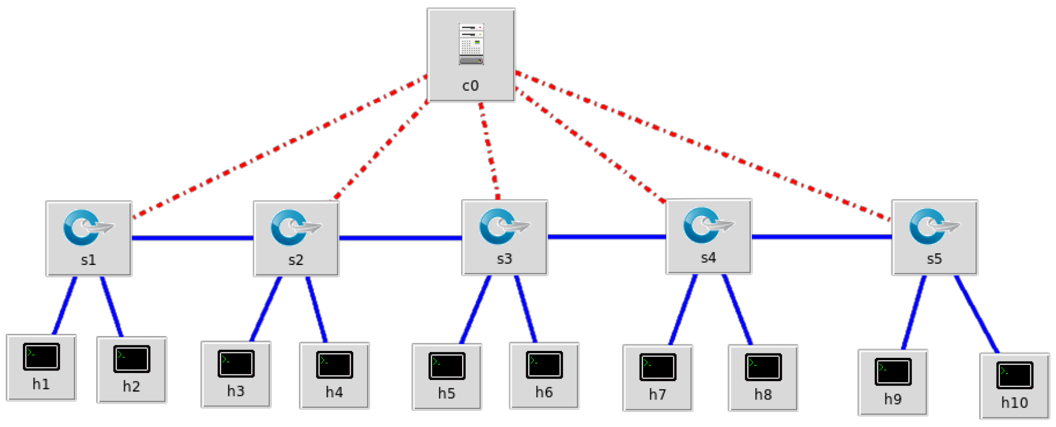


FIGURE 4.8: Mininet Topology for Experimental Testbed

We evaluated the feasibility and performance of the solution for three different network communication technologies such as WebSocket, gRPC, and REST. Finally, all results obtained in the testbed are an average of 30 runs that exhibited a limited variance of under 5%.

4.6.3 Benchmark of Network Communication Protocols

First, the testbed calculates the overhead introduced by the microservices-based approach in-terms of response time. To ensure this, we calculated three different latency: the response time of the first packet, the response time of the normal flow, and the average response time of rule updating packets. The response time for the first packet means, the time needed to send the first packet from one node to another. In an SDN network, with a reactive approach, the controller adds rules to the data-plane when it receives a new packet. This generates latency for the first packet of the flow. Once that is done, the flow can reach the destination node without passing through the SDN controller. Sometimes, during the flow, there are some updating packets to update, for instance, the expiration time of a rule. These packets generate latency because as in the case of the first packet, the flow must reach the SDN controller before. We repeated these experiments for different nodes of the network (H1 and H3 first, and then H1 and H10). As it can be seen from Figure 4.8, there are two switches between H1 and H3, and there are five switches between H1 and H10. To calculate these latencies, we sent a video streaming across the network and we kept the average round trip time. We repeated this test for each network communication technology.

The results, shown in Figure 4.9, show that the major delay is on the first packet latency. REST protocol appears to be seven times slowest than WebSocket technology, in H1 and H3 scenario, whereas gRPC protocol provides performance like the REST protocol but a little faster. The performance further degrades in the H1 and H10 scenario for all protocols. In particular, the REST protocol results around ten times slower than WebSocket technology. This is due to the multiple connections between switches. Finally, the performance of all protocols during the normal flow

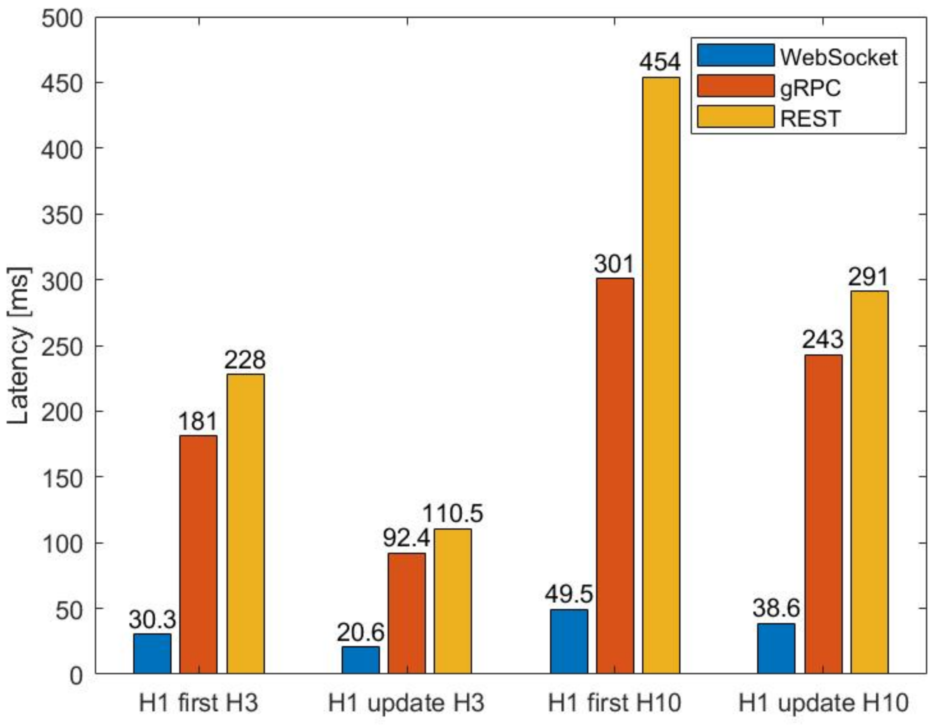


FIGURE 4.9: Performance results

was omitted due to the very low latency time (0.01 ms average around all protocols) but proves that all protocols are consistent and similar to each other.

In conclusion, we note that the REST protocol has a high response time for the first packet and rule updating packets compared to WebSocket and also to standard Ryu. However, the response time during the normal flow remains the same for all protocols. Therefore, it is apparent that the benefits of the microservice-based SDN model need to be balanced with any trade-offs incurred. On the one hand, despite the WebSocket protocol proves to be faster, it strongly depends on the Socket concept which means rely on the IP address and the Port number of the services. On the other hand, the gRPC protocol could become dominant in the future thanks to the adoption of the HTTP/2 protocol and to the use of Protobuf as the payload format. Furthermore, factors such as scalability and reliability (or availability) should be taken into account when deciding whether to use standard SDN or the microservices-based one. Moreover, the best choice of the right communication protocol depends on many factors including the context. For instance, a heterogeneous and ultra-reliable industrial scenario may require REST as a communication protocol to guarantee high connectivity among devices.

4.6.4 Resilience and Scalability Test

Finally, the testbed is designed to calculate the improvements of our solution in terms of the reliability and scalability of the system. To achieve these features, we used the ETSI MANO standard such as the open-source MANO implementation. The microservices-based approach allows the system to develop a horizontal scalable to easily adapt to the dynamics of the input workload and to tolerate potential run-time faults. Indeed, the OSM Autoscaling functionality automatically scales

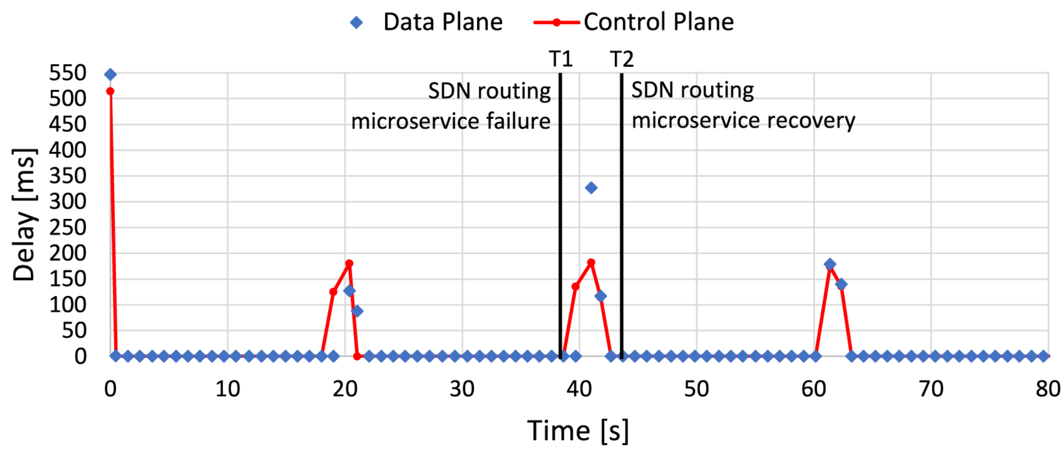


FIGURE 4.10: Resilience Test: Average Message Delay

VNFs based on available metrics such as CPU and memory consumption, packets received, packets sent, and so on. Each SDN component is a Docker container that will be encapsulated in a VNF. Therefore, there will be single or multiple VNFs that represent SDN applications or functionalities. As discussed before, the OSM Autoscaling function provides an automatic solution for fault tolerance management. This is possible through a scaling descriptor, that is part of the Virtual Network Function Descriptor (VNFD) which specifies the metrics and thresholds to be monitored.

Resilience Test

To test the resilience of the system, a node sends a continuous flow of packets. This test simulates the fault for an SDN routing path component and the possibility of instantiating a new instance as a backup. In this test, the communication between VNFs is via REST protocol and the MANO orchestrator helps to manage them by automatic scaling process. In particular we used a video streaming simulation, from a source to a sink node while the SDN routing microservice goes down. Figure 4.10 shows the average message delay recorded during the experiment. Each observation shows the delay in the control plane and the data plane. Since there is no message lost between time T1 and T2 but only small glitches in both the average message delay in the control plane and data plane at two instants prove the robustness of the system. The other small peaks correspond to the controller's rule updating packets at the datapath. The first delay is the delay introduced by the first packet, see the next experiments. This experiment shall be understood as a way to highlight the robustness achieve by the SDN system in its microservices-based deployment. Therefore, the standard SDN Controller, if some issues occur, is not able to react unless it is used in a distributed way. However, this means having two o more SDN Controllers deployed at the same time.

Scalability Test

We evaluate the scalability of the system in two different scenarios: multiple network service, and single network service. ETSI MANO defines network services (NSs) as a composition of VNFs that specifies a service such as an SDN controller. The first scenario - multiple NS - relates to the scalability of the entire NS that is comparable to a distributed SDN scenario. Figure 4.11 shows the related scenario

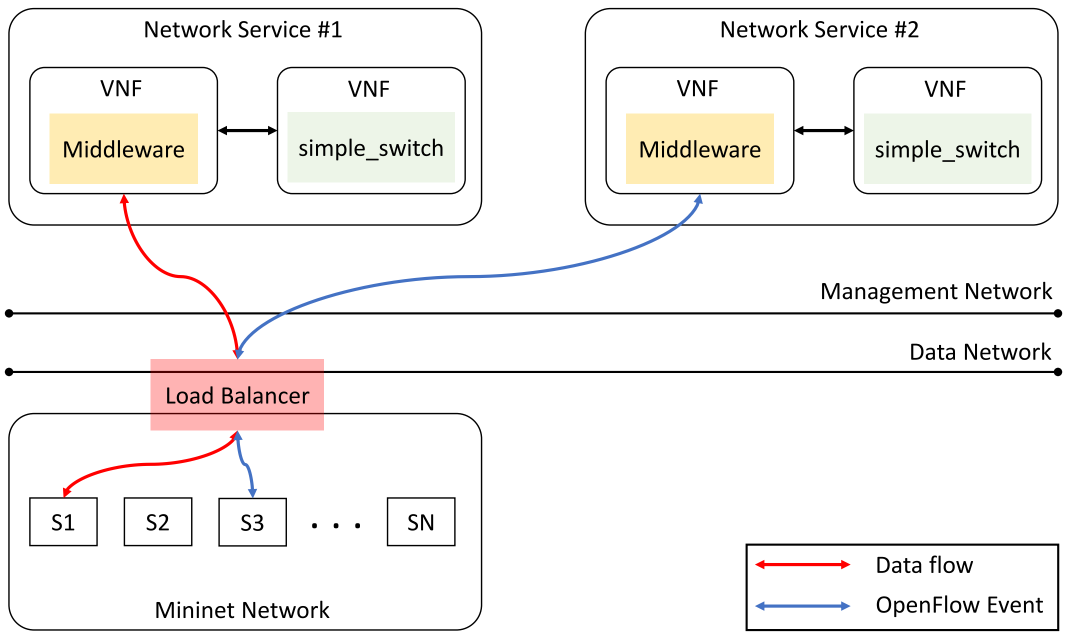


FIGURE 4.11: Scalability test: scenario 1 (multiple NS)

in the OpenStack and Open Source MANO platforms. The OpenStack provides a management network and a data network to correctly connect entities to each other, in particular, the Mininet network and the SDN system. In this scenario, we instantiate different NS to provide scalability. The load balancer in the mininet network provides a distributed control plane where each controller is in charge of a sub-set of the switches.

In the other scenario - single NS - we simulate the scalability of VNFs for instance the routing capabilities of the MSN prototype by keeping a single NS. Figure 4.12 shows the related scenario in the OpenStack and Open Source MANO platforms. The SDN system is composed of a single NS in which each microservices (as VNFs) can be instantiated multiple times to provide robustness and scalability. The autoscaling feature of the Open Source MANO allows to define the scaling descriptor as a part of the VNF definition. In particular, is possible to define several metrics to monitor and a load balancer (included in the middleware) that redirects requests to the routing service following a balancing strategy such as Round Robin and so on.

We calculated the average latency time for the first packet and for the normal flow by considering different host at different distances. The reference topology shown in Figure 4.13 is composed of 16 hosts and relates to a hierarchical network topology that is the most widely used in real data centers [136]. In the Figure 4.14 we show the average latency time for the first packet in both scenarios with 2 and 3 replicas, while the average time for normal flow is depicted in Figure 4.15. In conclusion, we tested two different scenarios in the OpenStack and Open Source MANO for the scalability concern, and we noted that replicating the entire NS performs better than a single NS. This because in our test the middleware is the bottleneck for the switches while splitting the management of the network to more middleware is more efficacy. However, in the second scenario, it is possible to replicate the middleware as well.

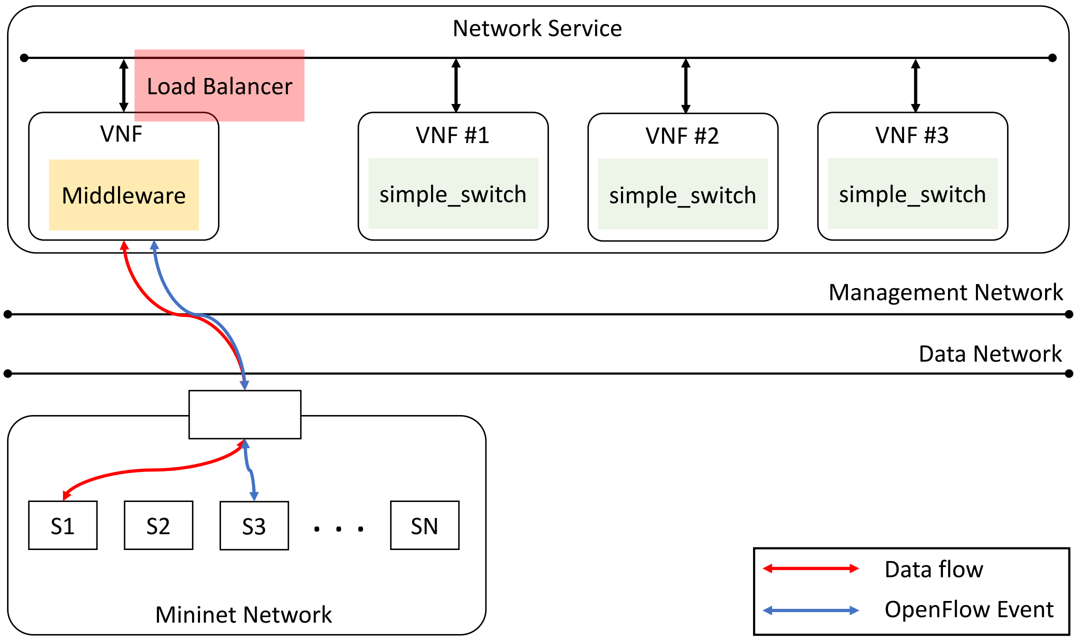


FIGURE 4.12: Scalability test: Scenario 2 (Single NS - multiple VNF)

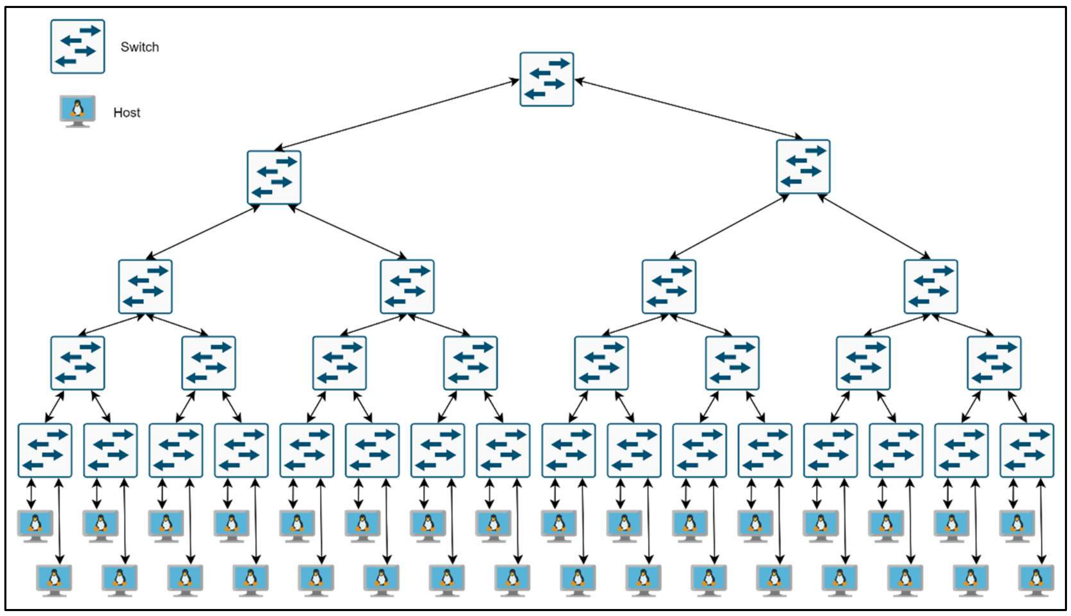


FIGURE 4.13: Scalability test: Network Topology

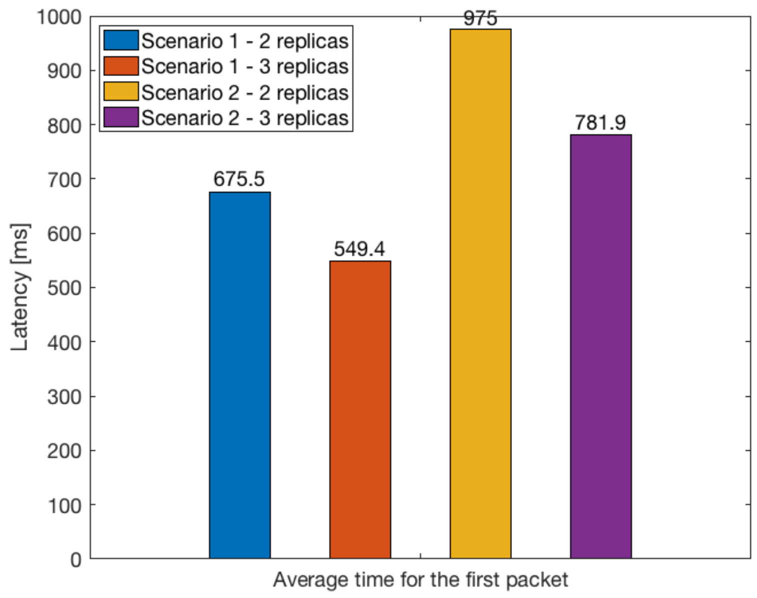


FIGURE 4.14: Scalability test: Average Latency for the First Packet

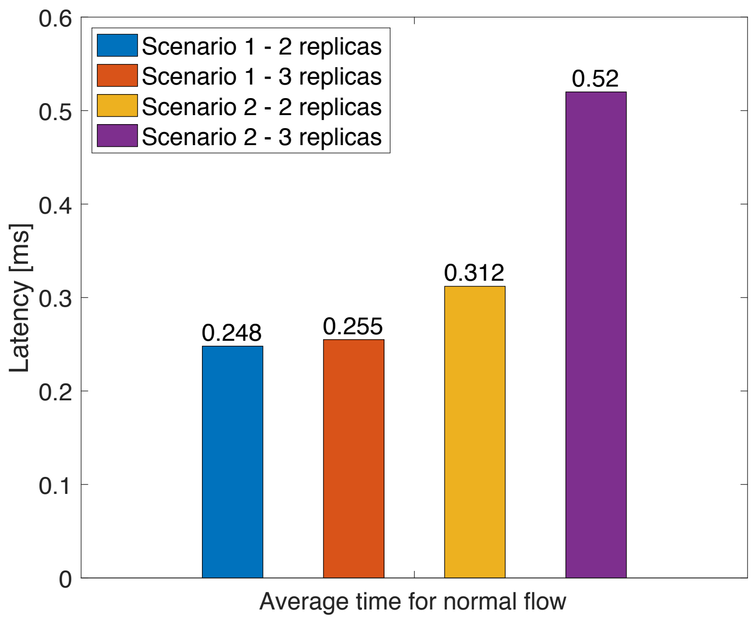


FIGURE 4.15: Scalability test: Average Latency for Normal Flow

Chapter 5

MANA-NMS based Unified Architecture and Its Application in 6G and Next Generation Network Management Systems

5.1 Introduction

The advent of 5G and the design of its architecture has become possible because of the previous individual scientific works and standardization efforts on cloud computing and network softwarization. SDN and NFV started separately to find their convolution into 5G network architecture. Then, the ongoing design of the future 6G network architecture cannot overlook the pivotal inputs of different independent standardization efforts about autonomic networking, service-based communication systems, and multi-access edge computing. In this context, this work provides the design and the characteristics of an agent-based, softwarized, and intelligent 6G architecture, which coherently condenses and merges the independent proposed architectural works by different standardization working groups and bodies. This novel work can be a helpful means for the design and standardization process of the future 6G network architecture.

The standardization of 5G started in 2016. The International Telecommunication Union (ITU) Telecommunication Standardization Sector (ITU-T) Focus Group on IMT-2020 concluded its pre-standardization activities in December 2016. The core reform that 5G introduced in communication was the idea of virtualization, or more specifically, network softwarization. This has also implied a tremendous paradigm shift from the previous store-and-forward to the current-future compute-and-forward. This has made computing as important as communication in future communication networks [92]. However, 5G did not come out abruptly. The design of its architecture and core characteristics has leveraged the previous experience, obtained during the successful development of cloud computing and network virtualization instances such as SDN – led by the no-profit consortium Open Network Foundation (ONF) and NFV started by the industry and standardized by the European Telecommunications Standards Institute (ETSI).

Recently, ITU has started writing a report about the future technology trends towards 2030 and beyond, which is going to be released in June 2022. This report will mainly provide the very general vision for future 6G communication networks. Then, in the last few years, speculations have been proposed to start shaping future 6G architecture, by research, industry, and standardization bodies. In this context,

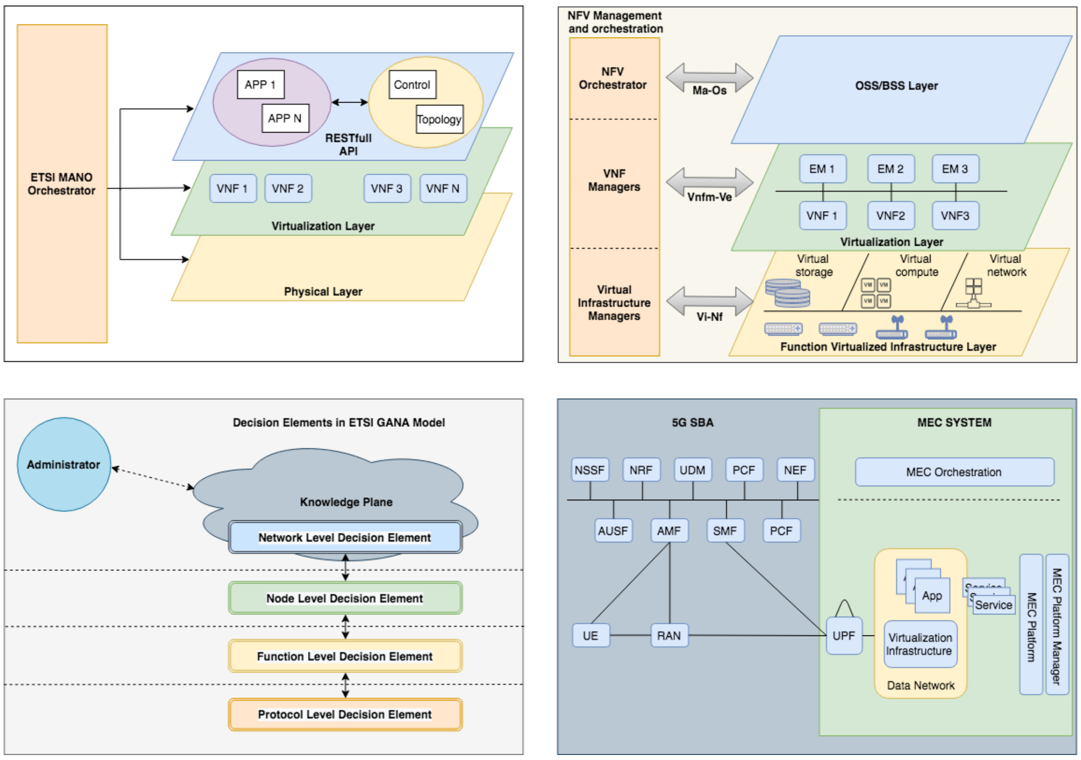


FIGURE 5.1: Architectures of SDN decomposed control plane (top left), ETSI SDN-NFV MANO (top right), ETSI GANA (bottom left), and 3GPP-ETSI MEC (bottom right).

among the core technical drivers of 6G architecture, there have been complete softwarization and modularization, and intelligence [115]. As it has been mentioned for 5G, this is also not coming out of the blue. The last few years of research and standardization efforts in these areas have now given the mature input for the envisioned architectural evolution towards the next generation.

In this context, this article provides an architectural design and guidelines for future 6G networks by leveraging the various standardization works by ETSI and 3GPP, considering the current popular trends on microservice- and agent-based intelligent communication systems. In this vision, all softwarized network functions become atomic entities, playing as autonomous, intelligent, and collaborative agents. To the best of the authors' knowledge, this is the first architectural attempt to unify the four existing individual architectures, microservices-based SDN control plane, ETSI SDN-NFV Management and Orchestration (MANO), ETSI Generic Autonomic Networking Architecture (GANA), and 3rd Generation Partnership Project (3GPP)-ETSI Mobile Edge Computing (MEC), namely a unique and homogeneous framework for 6G.

5.1.1 Motivation and Background

This section provides an overview of the main existing independent architectures adopting microservice-based communication systems, autonomic networking, and in-network intelligence.

5.1.2 Microservices-based SDN Controller

The centralized nature of the SDN control plane leads to several issues ranging from latency constraints to fault tolerance and load balancing [21]. These issues have been partially overcome by using the distributed SDN controller approach. However, the SDN controller is a monolithic system that results inefficient in several scenarios because it is costly to have replicas of SDN systems. Moreover, monolithic SDN deployment does not allow dynamic management of SDN components and/or functionalities to (de-)activate according to the scenario.

To alleviate this, the SDN controller can be decomposed and implemented as a set of software components running in a distributed fashion. In particular, it is possible to design the SDN controller as a composition of logical sub-functions, i.e. microservices. They can share network service load and creating a robust system against failures. The possibility of decomposing the monolithic SDN controller and designing the controller as loosely coupled components provides the possibility of a flexible controller deployment. That represents a step towards the cloud continuum and is in line with the vision of the edge/cloud hybrid architecture [40].

Accordingly, some research efforts have started to decompose an SDN controller into microservices. For instance, the Open Network Operating System (ONOS) proposes µONOS [203], which is the next-generation SDN architecture. However, µONOS has been specialized mainly for cloud datacenters scenarios and is limited to certain technologies, not all 5G compliant. In particular, µONOS uses gRPC protocols family for microservices intercommunication which is not the standard declared in the 3GPP white-paper v15 [1], which specifies REST APIs as the standard-de-facto for services intercommunication.

In this regard, we have started implementing a microservices-based SDN controller deployment that complies with the 5G standard, see Figure 1 (top left). Our implementation based on Ryu SDN Framework is still under development, but it is available to the community at the link: `https://gitlab.com/dscotece/ryu_sdn_decomposition/`.

5.1.3 ETSI SDN-NFV MANO

ETSI released it's ETSI SDN-NFV MANO architecture [92], to provide a unified architecture effectively combining SDN and NFV characteristics. Figure 5.1 (top right) depicts its structure. This architecture consists of three main entities, called the Network Management System (NMS), the Network Function Virtualization Infrastructure (NFVI), and the Operation/Business Support Scheme (OSS/BSS). The first manages the virtual network, the second set the resources (hardware or software) that are used to run and to connect virtual network functions, and the third sets the applications used by service providers to provide network services. Each one of these layers has an interface to the MANO entity, which hosts the virtual infrastructure managers (VIMs), that aim at controlling the NFVI resources. Next, the virtual network function manager (VNFM) configures and manages the life cycle of virtual network functions in its network domain. Finally, there is the orchestrator for the NFVI who manage the resources across different VIMs, and subsequently the life cycle of network services. Finally, the virtualization layer groups all the element management entities with their respective virtual network functions (VNFs).

5.1.4 ETSI GANA

Generic Autonomic Network Architecture

ETSI unveiled a standard reference architecture as an implementation guide for GANA architectural framework for network automation [177]. The main goal of the GANA reference model is to prescribe the design and operational principles for Decision Elements (DEs) as the drivers for cognitive, self-managing, and self-adaptive network behaviors. ETSI is performing several GANA instantiations onto various target standardized reference network architectures. This is to enable autonomic algorithms to be integrated into the design and implementation of DEs. The integration also aimed at standardizing the autonomic network architectures at four levels as depicted in Figure 1 (bottom left). These are:

- Network Level Decision Element: this level is designed to operate the outer closed control loops on the basis of network wide views or state as input to the DEs' algorithms and logics for autonomic management.
- Node Level Decision Element: this level is in charge of controlling the behavior of the Network Element (NE) as a whole, and also managing the orchestration and policing of the Function Level Decision Elements. GANA Function Level specifies the following four decision elements: security management, fault management, auto configuration and discovery, resilience, and survivability.
- Function Level Decision Element: represent a group of protocols and mechanisms that abstract, as atomic units, networking or a management/control function. The GANA model defines the following six Function Level Decision Elements: routing management; forwarding management; Quality of Service management; mobility management; monitoring management; service and application management.
- Protocol Level Decision Element: this is the lowest level decision element in the system. This kind of element is protocols or other fundamental mechanisms that may exhibit intrinsic control-loops or decision element logic and associated DE—as is the case for some protocols such as Open Shortest Path First (OSPF). These can be considered an example of the instantiation of a protocol-level decision element.
- GANA Knowledge Plane (GANA KP): enables advanced management and control intelligence at the Element Management (EM), Network Management (NM), and Operation and Support System (OSS).

A close look at the GANA model reveals the close alignment with the microservices-based approach. In other words, it defines the network functions as services considering them as an application that can be deployed in a virtual environment or container.

Multi-Agent System

Multi-Agent System (MAS) is a sub-branch of distributed artificial intelligence, where it has multiple interacting intelligent agents solving problems. Multi-agent systems include different attributes such as architecture, communication, coordination strategies, decision making, and learning abilities. We identified a multi-agent system as a competing candidate for defining atomic and autonomous DEs in the GANA model. DEs could be network function units that could be used as a building block in any automated system. Network function atomization is the mechanism that defines the smallest possible network function units in a service-oriented system. Multi-agent is comparable to microservices but with more autonomy and proactive capability. A comparative analysis and use of microservice and multi-agent are presented in [16].

5.1.5 3GPP-ETSI MEC Architecture

The basic idea of MEC is to provide capabilities closer to the end-users to overcome mobile difficulties. This promotes a new three-layer device-edge-cloud hierarchical architecture, which is recognized as very promising for several application domains [179]. With the advent of 5G networks, the MEC is one of the key technologies for supporting ultra-reliable and low-latency communications.

The 5G system architecture specified by 3GPP has been designed to fit a wide range of use cases including IoT networks management and tightly constrained scenarios. In particular, the 3GPP defines the Service-Based Architecture (SBA) for the 5G core network, whereby the control plane functionality and common data repositories are delivered by way of a set of interconnected Network Functions (NFs), each with authorization to access each other's services [266]. The SBA framework provides the necessary functionality to authenticate the consumer and to authorize its service requests, as well as flexible procedures to efficiently expose and consume services. Even the ETSI MEC defines an API framework aligned with the SBA framework to provide a set of functionalities and services. The resulting integrated architecture described in the white paper [266] is presented in Figure 5.1 (bottom right).

The complete *softwarization* of network functions and applications allow the unified management of these microservices by a standard orchestrator, such as MANO. This allows independent implementation and component reuse. Moreover, NFs could be deployed flexibly while performing functions autonomously and applications (i.e., services hosted at the edge) can belong to one o more network slices.

5.1.6 Proposed Unified Grad Architecture

This section discusses the unified architecture, providing conceptual analysis.

5.1.7 Proposed Unified Architecture

This section presents our novel fully unified architecture for automated network systems. As indicated before, our architecture combines four different standards namely, SDN, ETSI NFV, ETSI GANA, and ETSI MEC, and intends to propose virtualized network agents instead of virtualized network functions. The definition of an agent and its design is discussed in the next subsection. Figure 5.2 shows our unified architecture reporting its general structure. While the infrastructure and virtualization layers are directly derived from NFV architecture, the MANO layer is modified so that the internal components of the MANO are defined to be intelligent orchestration agents. The application layer of NFV is significantly modified and contains the four layers of the ETSI GANA model and its decision elements defined to be agents. Therefore, the modified ETSI GANA divides the application layer of NFV into protocol level agents, function level agents, node-level agents, and network-level agents.

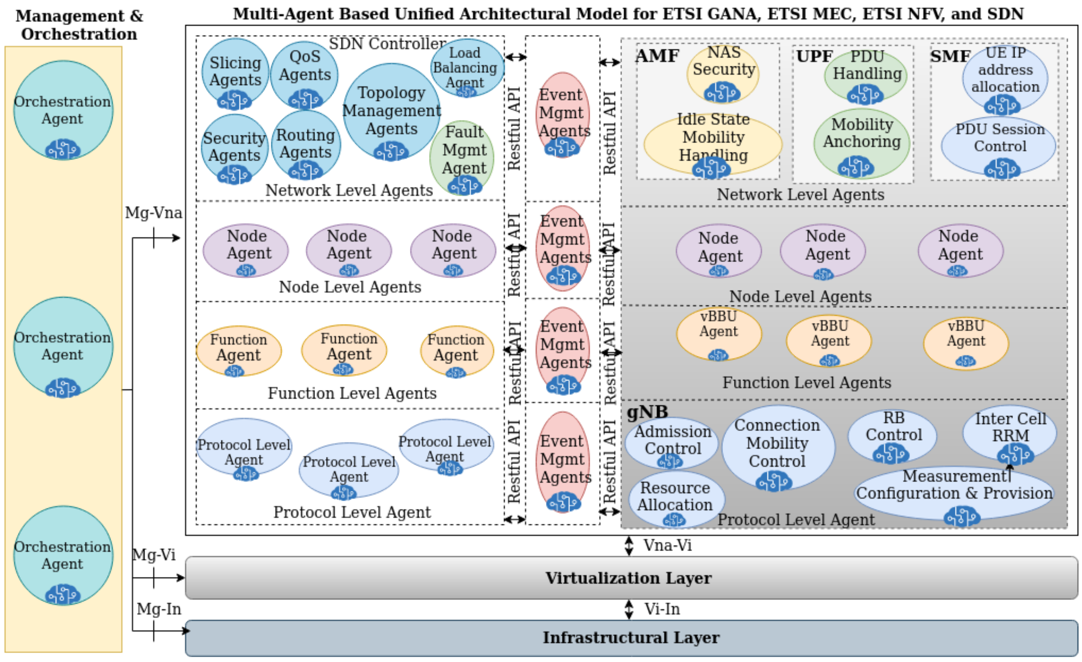


FIGURE 5.2: Microservices and Multi-Agent Based Unified Architectural Model for ETSI GANA, ETSI MEC, ETSI NFV, and SDN

The event distribution layer allows the interrelationships among the components of the unified architecture and the SDN controller. Note that for event management several strategies are available including centralized, distributed, and hybrid [277].

The application layer is also divided vertically showing the decomposed SDN controller on the left side. The functions are built as autonomous and atomic agents. The interconnection of these agents would create the required SDN controller functionality depending on the requirements in a given environment. The final SDN controller system becomes a multi-agent system. The same could be said about the wireless functions 5G and 6G networks where functions are designed as autonomous agents and the collection creating the full wireless functionality.

The interface between the infrastructure layer, virtualization layer, and the application layer is the same as in NFV; however, the orchestration layer is based on agents. Therefore, we recommended interfaces to be open and adaptive based on the specific applications. This could be achieved by equipping agents with a Programmable Protocol Stack (PPS), which represents the implementation of a software-based environment of network protocols and layers [27]. In the currently proposed scenario, the interface between the agents is mainly through restful API. However, as will be discussed shortly, we suggested equipping the agent with a dynamic protocol stack.

5.1.8 Network Function Atomization

First, let us define the concept of network function atomization. Network functions can be divided into atomic units to allow for maximum re-composition freedom. The concept of network function atomization is first discussed in [44], where the authors presented the idea for Wireless Sensor Network (WSN) node organization. As per the original concept, every node in the WSN is equipped with an autonomic element which is the smallest object in an autonomic network. Besides, the idea in [8] was a considering the composition of autonomic elements. This composition is

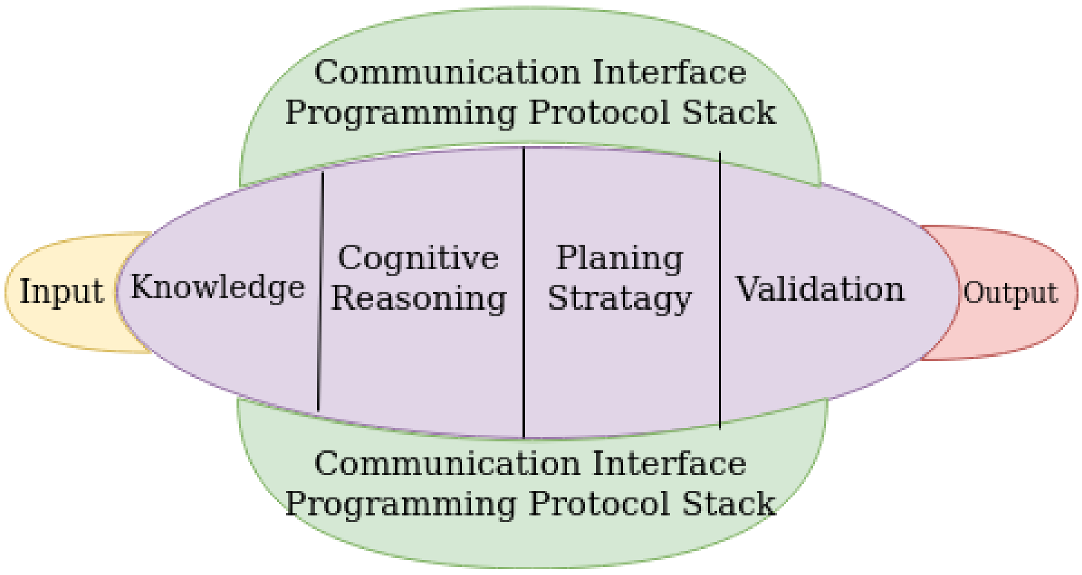


FIGURE 5.3: Agent Internal Architecture

intended to include an autonomous overlay network management structure and a self-organizing composition against autonomous overlay networks.

Here we re-define the atomic units as a network functional unit with the full capability to perform a given function such as resource allocation, QoS monitoring, etc. In the proposed architecture, we define the service or internal components of a decomposed SDN controller to be agents. Since there is no universally accepted definition for what is an agent, we first provide a basic definition of what is an agent from our context.

5.1.9 Agent Internal Architecture

As depicted in Figure 5.3: an agent is composed by an *INPUT*, which is an incoming request; *FACTS*, are the knowledge database of the agent; *COGNITION (REASONING) UNIT* gives agents the reasoning capability; *PLANNING STRATEGY* organizes the steps or procedures for the action to be taken to satisfy the requested service; *VALIDATION* is the unit that verifies the action plan for consistency; and an *OUTPUT/ACTION* is the final decision (result or action) to be taken by the agent. An agent also has an interface to communicate with other agents or microservices via the PPS.

Future generation networks should be equipped with effective and efficient protocols. Agents in the application layer of the proposed architecture are various types that will use combinations of multiple protocols for communication for various kinds of applications including real-time applications. A programmable protocol stack is a software-based stack that inputs information from the higher logical layers and configures various parameters at each layer, repeating the procedure at each network device. This update of parameters is dynamic according to network changes and end-to-end service requirements.

5.1.10 Multi-Agent Based SDN Controller Decomposition

It is possible to design the SDN controller as a single monolithic process, as a confederation of identical processes arranged to share the load or protecting each other

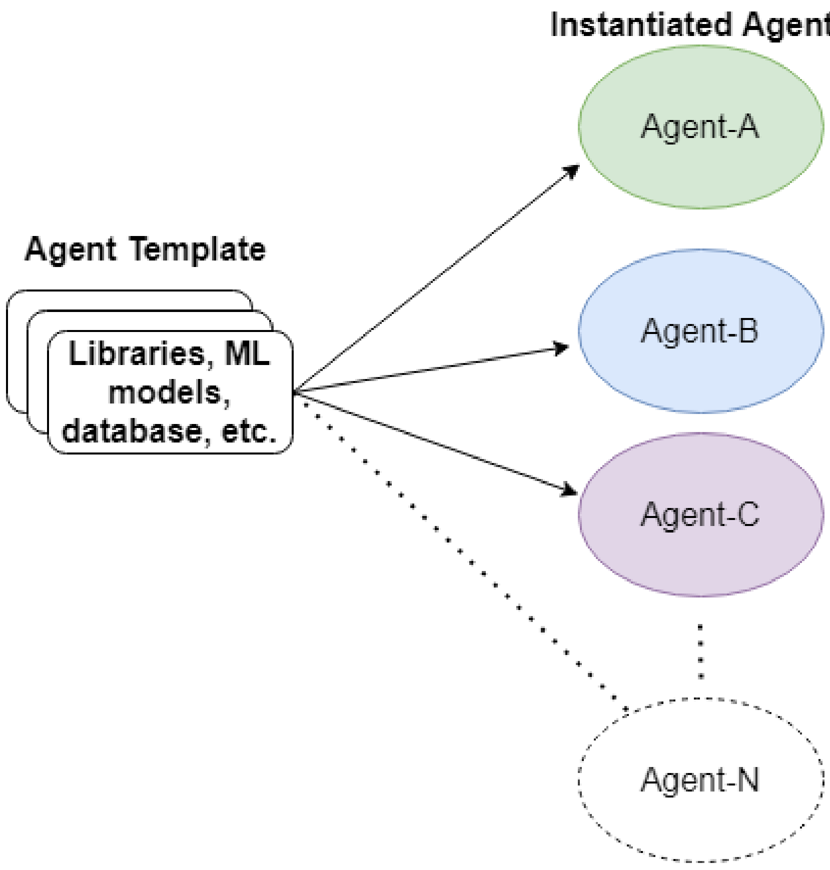


FIGURE 5.4: Agent Instantation From a Template

from failures, or as a set of distinct functional and collaborative components. Moreover, any combination of these alternatives is possible. Decomposing the SDN controller, designing the components as autonomic agents, and recomposing them to create the SDN controller is a big advantage in the era of functions' *containerization*, and *cloudification*, where features of loose-coupling and distributed deployment are required [100]. The main principle to retain in decomposing an SDN controller is that the network information and state should be synchronized and self-consistent. This allows an independent implementation and component reuse. Instead of using microservice, the proposed architecture re-define every SDN controller component as an atomic agent. Controller's decomposed components can be executed on arbitrary computing platforms on distributed and virtualized resources such as virtual machines (VMs) or containers in edge or cloud data centers. The loosely-composed system can be viewed as a black box, defined by its externally observable behavior, emulating the original monolithic SDN controller. The agents could be orchestrated by a standard orchestrator, such as MANO, creating an agent chain to equivalently perform what the legacy SDN controller does. In particular, the controller could be deployed flexibly while performing the function autonomously.

Chapter 6

Future Application of MANA-NMS

6.1 MANA-NMS for IoT Connectivity Technology Management for Space Application

6.1.1 Internet of Things

Internet of Things (IoT) applications are bringing immense value into our lives touching ranging areas such as health monitoring, green energy, environment monitoring, smart home, and smart city[228, 304, 6, 249]. With newer wireless networks, superior sensors, self-healing soft-electronics, and revolutionary computing capabilities, the IoT is already the frontier in the race to explore and provide technological advancement in the livelihood of a human being. Due to this, IoT has caught the attention of researchers and private industries. The rate of interconnected IoT devices is overwhelmingly increasing and continuously growing with time. As more IoT objects are connected, there is an increase of information in the form of data in the interconnected system[247].

IoT for space application is starting to take shape. Currently, IoT in space is at the conceptual development stage than actual applications. It is because of many obstacles to overcome before organizations can start to deploy and use IoT in space for practical applications. However, a different and alternative approach may need to be explored. Spacial on-site manufacturing and utilization of IoT devices are more viable than transporting them from the earth over a long distance. Both mechanisms have huge challenges before being realized and are sometimes complementary in that what can not be manufactured or important to initial materials should be transported. What can be manufactured in remote sites could help the exploration paving the way for human transportation and presence preparing for human arrival. This article reviews the most recent research activities on the application of IoT technology for space applications.

This challenging issue is difficult to resolve with the existing infrastructure. It means that there should be a solution with a new concept and approach that takes data rate, performance, and physical environment into consideration in trying to come up with interplanetary communication. When communication and controlling technology are advanced enough, IoT is expected to have a huge potential to revolutionize space exploration. The peculiarity of space exploration which comes due to the vast distance to the target environment to be studied has tremendous challenges. The challenge is to effectively deploy, configure, control, and manage remotely which requires extremely expensive operations.

NASA is putting an incredible effort into the adoption of IoT for space application. It has already setup an IoT lab[129] at Johnson space center and other virtual labs in anther places such as at Ames research center, Kennedy space center, and jet propulsion laboratory. It was setup in the federal government, which has completed the first phase and documentation, searching for an IoT platform and collecting data on the twenty selected devices.

More recently, in another effort, NASA and Stanford collaborated to launch a tiny IoT satellite into Earth's Orbit"[256]. NASA named the centimeter-scale satellites sprites or ChipSats. The main purpose of the IoT satellites is to perform research activities. More than hundred of them are already in orbit by the spring 2019. First confirmation signals had been received the back by next day. By enabling communication between the satellites, they would like to demonstrate how the satellites can work together. This is necessary if they eventually operate in a swarm.

The launching of TechEdSat-5 nanosatellite, which is a Technical Education Satellite-5, is a specific example of the application of IoT in space[273]. The TechEdSat-5 nano-satellite is a 3U CubeSat which sometimes alre called as TES-5. It is developed by students of San Jose State University, the University of Idaho, and NASA's Ames research center. The main objectives of the TES-5 are to establish an improved uncertainty analysis for eventually controlled flight through the thermosphere. It performs a detailed comparison to the TES-3 and TES-4 concerning key thermosphere variable uncertainty. It also improves the prediction of re-entry location while providing the base technology for sample return technology from orbital platforms. Moreover, it provides the eventual testing of independent TDRV-based planetary missions. Furthermore, it provides engineering data for an on-orbit tracking device that could improve the prediction of jettisoned material from the ISS[124].

Lander to mars-rover communication may require better connectivity in terms of QoS, latency, reliability, and range. Whereas, for environmental monitoring requirements, it could be satisfied with unlicensed LoRa-based IoT devices for environmental measurement parameters, such as temperature, humidity, soil content, and so on. Moreover, the connectivity between the two technologies could further increase the possibility of more types of device interconnection at various locations of the planet and times of the mission. This gives the mission further possibility of exploring more information about the target planet in a single mission. Moreover, the same mission may have single backbone connectivity from landers to the geostationary satellite station or directly to earth stations. These with the interoperability of IoT networks, the collected information using various technologies could be forwarded through a single interconnection point. Moreover, this enables the processing of each data collected from each type of IoT device at some aggregation point such as edge computing. The processing of the collected data would reduce the size of integrated data for efficient transmission. Interoperability enables this possibility.

Edge computing has also seen its way in space demanding a new way of designing and transporting satellites. The challenges and functions of edge in space application demand are presented in[118]. Many IoT applications have requirements that cannot be met by the traditional cloud[245].

Space is not free from adverse competitive animosity which could result in security concerns between major space players in the race for space exploration. IoT devices implanted in space for measurement and other exploration activity could be attacked or hijacked by the adversary. Unattended access or hijacking of a single IoT device or robot or wireless sensor network may result in unintended consequences, in which the sensors are placed on territory accessible by adversaries. Therefore, the security mechanism for IoT devices and connectivity networks is of the essence. In [89], the authors discuss adaptive feedback-supported communication for IoT in space application. The suggested technique is to minimize the amount of transmission from the wireless sensors making the task more difficult or impossible for the adversarial observer. In this sense, it is to take advantage of hiding the sources of wireless communication. Moreover, the technique has a further advantage in that any decrease in the number of transmitted signals from the source node can also contribute to energy savings. This can prolong its operational time giving by minimizing the energy expenditures for communication from a spacecraft or wireless sensor node. In[234], a security framework is provided which is intended to provide support to IoT device producers. The author proposed a framework called IoT-HarPSecA (A Framework and Roadmap for Secure Design and Development of Devices and Applications in the IoT Space) [234]. IoT-HarPSecA offers three main functional features: security requirement elicitation, security best practice guidelines for secure development, and a feature that recommends specific LightWeight Cryptographic Algorithms (LWCAs) for both software and hardware implementations.

6.1.2 Network Coverage, Network Softwarization, and Network Automation

Network connectivity on remote sites such as mars and the moon and communication with earth are the main aspects that we will consider in this section. There is a great advancement in the networking industry that is implemented as well as under development to be deployed in the near future. These technologies are providing a tremendous benefit in various sectors of human development. These are from simple voice communication to the video call services, from simple computer interconnection up to the worldwide web from simple on-demand video access to the critical for remote surgery. The advancement has enabled various types of devices to interconnect providing worldwide coverage with various types of technologies wireless and wired. To enable this various mechanisms are developed such as twisted pair cable, coaxial cable, and fiber access as a physical transmission. The same is true in the wireless domain. Several (de)encoding, channel access protocols, (de)encryption techniques, security tools have been developed to enable communication through both wired and Wireless. Numerous architectural models, theoretical concepts, implementation mechanisms, have also been developed and continuously improved. In this subsection, we review important advancements in networking that could have the potential to be adopted in space exploration.

Backbone Network Technologies for Space Applications

The existing communication between moon/mars and earth is through wireless links. For example, the Curiosity rover, which had touched down mars, sends radio waves through its ultra-high frequency (UHF) antenna with 400 Mhz to communicate with Earth through NASA's Mars Odyssey and Mars Reconnaissance Orbiters. To serve as both its "voice" and its "ears.", curiosity has three antennas. They are located on the rover equipment deck, at its back. Having multiple antennas provides backup options. There are networks of antennas deployed in three strategic locations of the earth. They are called Deep Space Network (DNS) which are located in the United States (California), Spain (Madrid), and Australia (Canberra). They support NASA's interplanetary spacecraft missions[299]. Each DSN site has one huge, 70m diameter antenna. The antennas are the largest and most sensitive DSN antennas. They are capable of tracking a spacecraft traveling tens of billions of km from Earth [73, 299].

There is some advancement in satellite-based networks that could be extended to encompass deep space communication. This technological advancement and convergence of satellite communications would provide a converged network of networks such as a worldwide web in mars, moon, earth, etc. In [308], the authors suggested a potential architecture of Space-Terrestrial Integrated Network (STIN) that integrates the existing Internet, mobile wireless networks, and the extended space network. The architecture is aimed at providing comprehensive services globally that can be accessed anytime and anywhere.

In this regard, the backbone plays a crucial role in interconnecting geographically distributed and vast distant networks. On earth, the transitional backbone network extends 100 to 1000 km distances. However, when it comes to space the backbone network ranges in millions of km. Thus it is significantly affected by the distance in terms of electromagnetic wave propagation and physical deployment possibility. The difficulty of erecting a wired technology for space communication hinders the use of traditional backbone technologies such as coaxial cable and optical networks. The most viable technology that could be used as a backbone network is wireless communication. This could be through radio links as in the case of DSN, microwave links, and free-space optical networks.

Network Coverage Technologies on Remote Environment

Network coverage in the remote site could take some inspiration from the existing technology that is implemented on earth[246]. The most convenient connectivity technology that could have an important contribution to space exploration is wireless technologies. For example, a wireless cellular network could be used to provide a wireless access network on mars[231, 68]. The authors in [68] discuss the possible use of IEEE 802.11 a and b wireless local area network (WLAN) standards for proximity wireless networks on the Martian surface. They presented modeling of the physical layer. Moreover, in [231] discussed the communication aspects of Martian missions, based on the deployment of a Martian wireless network infrastructure using LTE on Mars (LTE-M). Other existing works in the area of access coverage through cellular, drone and balloon-based network coverage could be considered to adopt in Mars[248]. The physical layer modeling of the Martian surface is dependent on the Maritain atmosphere and terrain. Depending on the geography of mars, it also varies from place to place that should be considered in the design and modeling of the physical layer signaling propagation.

Depending on the mission plane, which could be long term or short-term plane, the technological adaptation in providing the coverage could also be considered. Dynamic changes as the exploration mission being executed the technology needs changes over time. E.g., first the mission could be to evaluate the composition of a given place and weather conditions of the same place and time. In that scenario what kind of device should be used, and what kind of rover should perform the task should be defined. Based on the required exploration task the network could be dynamically provided. Moreover, when the mission changes, which could be to check on the other part of the martian surface such as Eberswalde, Holden Crater a different network dynamics could be configured that could be based on drones or balloons.

The work in [284], presents initial results concerning modeling the RF environment on Mars in support of determining the characteristics of potential wireless, rovers, and sensor networks. The work utilized commercial RF propagation modeling software, designed for cellular telephone system planning, together with recent topographic data for Mars to determine ally construct propagation path, loss models.

A code division multiple access communication system for Mars based on geostationary relay satellite is presented in [19]. The paper defines CDMA based communication system between various assets such as rovers/landers on Mars surface, low Mars orbiters such as MRO, CubeSats. They are in the vicinity of Mars, and they use a geostationary relay satellite at 17,000 km above Mars's surface. Using 8.40 GHz frequency, the assumed data rates are between 50 Kbps and 1 Mbps. In[89] proposed an adaptive feedback-supported communication technique that can minimize the energy expenditures for communication from a spacecraft or wireless sensor node.

There are various IoT connectivity technologies with the potential to revolutionize space exploration. The first IoT connectivity technology to be adopted in space is Wi-Fi. Wi-Fi® has enabled a networked space exploration. NASA has provided wi-fi access by installing the first access points (APs) on the International Space Station in 2008[300]. Lora could also be the next to provide connectivity in mars or moon exploration.

Virtualization and Softwarization

Virtualization and Softwarization of the network will help tremendously for two main reasons: reduction of the need for physical equipment and generalization of the hardware required (general purpose CPU, Memory, and storage).

Software-defined networking(SDN) is a centralized and programming of networks through a centralized controller. This provides flexibility and dynamic controlling of networks. In remote exploration like that of mars, SDN is an ideal approach for network operation and management. It provides the possibility of developing a dynamically adaptive network. Moreover, it paves the way for the autonomic controlling of a network through artificial intelligence (AI). Network function virtualization (NFV) is also an important network technology that provides a software version of network functions that provide the controlling, management, and operation of the network.

A Software Defined Radio (SDR) is a softwarized radio communication system. It uses software for the modulation and demodulation of radio signals in a communication. In the future communication subsystems for space exploration missions could potentially benefit from SDRs controlled by machine learning based algorithms.

Network Automation

Automated network management is necessary for deep space exploration due to the difficulty of human presence in space. Network automation is the capability of the network to manage itself independently. Autonomic networking is required to scale up the network management capability to address the expected dynamic growth networks. Due to the obvious reasons for the unavailability of humans to install and manage the network, we require the following capability of a network that should be deployed on mars.

- Self-Installation:installing hardware equipment is required to provide coverage. Once the required equipment is delivered in the appropriate places, the network equipment has to install itself. The delivery could be through a martian rover or drone. The installation could require digging holes on the Marian surface to fix the antenna or other required hardware equipment. The digging, placement, and fixing of the hardware may need to consider the Marian surface for dust and rocky areas.

- Self-Configuration is the capability of a network to configure and re-configure itself based on predefined policies to achieve a given performance. This should happen seamlessly and with no human intervention.
- Self-optimizing means to ensure that the network always uses the available resources to provide the best possible performance even in highly varying environments. The network should always measure its current performance and set strategies to efficiently perform in case of any deviation from the set of expectations and predefined-ideal standards.
- Self-protecting is related to the security of the network and it is a very crucial issue. Self-protection ensures that the network can shield itself against any potential attacks such as Denial-of-Service(DoS) attacks.
- Self-healing is needed in case of failure of any network element. A network with self-healing capabilities is able to recover from such failures in the shortest time possible. The network is able to discover and automatically repair any failed elements to ensure service continuity.
- Self-drone based areal coverage which requires driving drone in unknown environments.

6.1.3 Artificial Intelligence for Space Applications

Artificial intelligence would play a significant role in the massive Martian exploration in a range of areas. This includes the automatic controlling of the navigation of rovers in the Martian surface; areal maneuvering of Martian helicopters for various missions; Controlling of networking management system; performing analysis of the collected scientific experimental data; automated manufacturing of equipment, tools, chemical products (e.g CO2), etc. Few works explore the application of AI for space exploration. For example, the authors in [90], presented a multi-objective reinforcement learning-based deep neural network for cognitive space communications. They presented a hybrid radio resource allocation management and control algorithm that integrates multi-objective reinforcement learning and deep artificial neural networks. By monitoring performance functions with common dependent variables that result in conflicting goals, they aim to efficiently manage communications system resources. Another interesting work is the data mining application of AI[112]. The authors presented an ML-based telemetry data mining of space missions. An application of AI in aerospace is presented in[145].

6.1.4 Cloud, Edge, and Fog Computing

Computing is required to perform various analyses on mars or any remote mission. This could be the weather condition soil content, chemical composition of rocks, analysis, and before sending them back. Even autonomic control of rovers, drones, and networks that provides coverage requires huge storage and computational power.

As demonstrated by the recent perseverance rover landing on Mars, it is possible to reprogram the onboard device for a different mission. For the perseverance, once the rover has landed, the computer is re-programmed by commands sent from Earth by NASA engineers to perform mobility visual processing. This demonstrates the possibility of complex task execution by a single rover or more collaborative rovers in the future.

However, for massive and complex missions it may need various rovers, drones, autonomous equipment, or other IoT devices. The collaboration of such a mission requires both network coverage and standard computing. It is possible to fully equip the collaborating devices with internally embedded computing. However, it will be inefficient in a distributed and collaborating mission. Therefore, a cloud-based computing provisioning to a remote mission will demonstrate significant efficiency in availing storage and computing power to the exploration missions.

Computing, networking, and control cannot be alienated in a space mission. Moreover, computing, control, and networking are complementary technologies that could facilitate the Martian massive exploration. Principles of control help for network control, edge computing for networking, networking for clustering, and interconnecting of separate computing units. Computing provides a resource for sophisticated controlling algorithm computation. An interesting work on Mission-critical control at the edge and over 5G network are presented in[251].

Teleoperation Using Edge Computing

Remote controlling of a networked system has been studied as discussed above. However, recent advancements in networking through network softwarization and automation, cloud computing, edge computing, machine learning, IoT, UAV, and automation have instigated the need for a new approach considering the current advancement in these cross multidisciplinary domains[250]. Moreover, the target of this literature server is space exploration martian and moon exploration in particular. Edge computing is an emerging paradigm aiding the responsiveness, reliability, and scalability of terrestrial computing and sensing networks. To alleviate the problems caused by the long distance between the processing platform and the terminal, edge computing provides a new paradigm for space applications. An interesting work on orbital edge computing is proposed in [75], presenting conceptual definition and characterization. They described power and software optimizations for the orbital edge. They also discussed the use of formation flying to parallelize computation in space.

Since the concept of edge computing in space is relatively new, there are few works on the deployment for deep space exploration. However, an application based on space edge computing is also discussed in [141]. The authors presented a real-time motion control method using measured delay information on access edge computing. Similarly, the work on optimal control design for connected cruise control with edge computing, caching, and control[293] is used for remote operation on earth. In the paper, considering the communication delays, an optimal control design is proposed for the system with edge controllers having the capability of computing, caching, and control. First, the dynamics of each vehicle in the platoon are modeled in detail, and then a linear quadratic optimization problem is formulated when the sampling period and the communication delay are considered. To minimize the deviations of the vehicle's headway and velocity, the optimal control strategy is iteratively solved using backward recursion. A survey on edge computing for the IoT is presented in [312].

A recent interesting work using satellite for edge computing for IoT in aerospace is presented by Wang and et.al [292]. They propose converting the legacy satellite into a space edge computing node. This enables to dynamically load software in orbit, to flexibly share on-board resources, and to provide services coordination with the cloud[245]. They also provided the hardware structure and software architecture of the satellite. The work in [297] discussed the application of edge intelligent computing in satellite IoT. Similar work with a focus on latency and energy optimization for MEC on enhanced SAT-IoT networks is presented in[66]. Interesting recent work for industrial remote control application is presented in [167]. The paper explored the use of edge computing for multi-tier industrial control system.

6.2 Summary and Conclusion

The first part presented an end-to-end comprehensive formulation of C-RAN system that deploys VNFs in an edge data center. Based on the formulation, we have analyzed the service throughput, end-to-end latency, end-to-end reliability, computational and energy consumption, and overall service admission. Moreover, we reformulated and analyzed our C-RAN system mathematical modeling, considering backup VNFs and computational over-provisioning techniques for reliability and latency improvement, respectively. Furthermore, we proposed service categorization and differentiation in applying the indicated techniques to improve resource utilization and power consumption. The performance evaluation suggested that reliability and latency improvements could be achieved while increasing the admission of service Type I. However, this is achieved with an increase in both resource utilization and power consumption having a direct implication on CaPex and OpEx. The same is for the admission probability on operators' total revenue. Therefore, the application of service differentiation provides a degree of freedom for the operator to optimize, considering CaPex, Opex and revenue.

We have also extended the work to provide a solution to tackle the network level interoperability problem in IoT connectivity technology. We have developed a network protocol for translator to be deployed as a virtual machine or a container. The translator can be instantiated within the Access Point on-demand, and de-allocated when it is no more necessary. We have also implemented the proposed approach implementing it using the NS-3 simulator as proof-of-concept. The proposed approach enables the flexible deployment of a standard translator. Moreover, the dynamic deployment provides efficient usage of the translator for applications with resource constraints. The performance evaluation shows the functionality of the proposed techniques. Moreover, we have evaluated the delay in the translation. As an emulation example, we emulated LTE, LTE-A and 5G deploying all the components of 4G/5G on a single laptop. It can be used and adapted to implement the more diverse and complex scenarios. The advantages of deploying 4G/5G as a cloud application are: an autonomous management of resources and services, a better migration of applications across different environments, a central control, etc. Beside autonomous reconfiguration that reduce time and costs, we have a centralized and efficient way to re-configuring components. This avoids errors due to manual handling of configuration files.

The second part focus on a network automation architecture using multi-agents. Network softwarization enables myriad innovative possibilities. The most important innovations of network softwarization are design flexibility and programmability. SDN and NFV architecture miss network automation. The advancement and explosion of machine learning would further enable the realization of full network automation. Autonomic networking is becoming the necessity for future networks, that are aiming at interconnecting billions of heterogeneous device, with stringent services' requirements. We first presented a review of autonomic networking since 2004, categorizing literature into before and after the introduction of SDN and NFV. We then discussed the challenges of network automation. Multi-agent based network automation architecture is presented along with the Markovian mathematical model of the system. Finally, we presented the performance of the system for a simplified case. The performance analysis showed the possibility of utilizing a multi-agent system for network management system automation. What we can guess from the preliminary analysis is that the use of multi-agent in automating network management provides a scalable, flexible, dynamic, and resource-efficient system. However, due to the distributed nature of the system, the effect of virtualization and communication overhead has to be evaluated as future work. Testbed implementation of MANA-NMS and the exact implementation of agents for specific network functions are also left as future works.

Moreover, we have also showed how to design an agent using ML as internal component of the agent. We integrated two main technologies, ML and MAS. To achieve network automation, these agents were charged with making decisions on behalf of the network. The brain of the NTCA designs considered were four machine learning algorithms: SVM, K-NN, Naive Bayes, and Decision Tree. We evaluated the best performing NTCA design based on performance metrics such as; classification accuracy, training latency, and classification latency. The results showed that the Decision Tree NTCA has the highest mean classification accuracy, the smallest mean training latency, and the smallest mean classification latency.

The third part of the PhD work focus on monolithic system decomposition into loosely coupled functions using the two comptting service oriented architectures, Microservice and Multi-agent systems. We first decomposed SDN controller using Microservices. Here a microservices-based decomposition architecture is proposed for the SDN paradigm to improve agility, scalability, and reliability. The SDN features including dynamic flow control and the possibility to reconfigure the network according to application needs make it an enabler for the 5G next-generation IIoT networks. However, most SDN controllers are deployed as a monolithic block, and that can make them not efficient enough to cover the requirements of the IIoT networks such as scalability and robustness.

The MSN framework paves the way to a new generation of microservices-based approaches for the next generation 5G-ready SDN networks. The use of microservices represents a big step ahead for the Cloud Continuum also in the vision of the Edge/Cloud hybrid architectures [40]. This paper inspects the use of microservices in the SDN paradigm presenting pros and cons of this novel paradigm when employed in this specific domain. The results presented in this paper are focused on the delay for the first packet, which means the latency introduced by the decomposition and distribution of microservices. Obtained results demonstrate the feasibility of applying microservices-based architecture to the SDN paradigm, which offers a wide range of benefits including deployment agility, scalability, and robustness that can be granted for each different SDN controller functionality. In the proposed solution, we take into account more ways to interconnect microservices with each other by analyzing different protocols such as REST, gRPC, and Websocket. That allowed to compare not only the introduced delay by the mentioned technologies, but also the benefits that each protocol could bring. The tests show significant improvement in terms of reliability of the system in the case of a microservice become unavailable. Moreover, tests on the scalability show how to achieve scalability on the MANO orchestrator with two different scenarios. Therefore, orchestrating microservices for managing fault tolerance or for distributing the load will be a task for the MANO orchestrator. We also provide to the community working in the field our implementation [1]

Boosted by obtained results, we are now working along different ongoing work directions. First, we are using a reactive approach in the evaluation, we are working on implementation also a proactive approach so to further improve some aspects of latency. In fact, a reactive approach allows a system to react when something happens, for instance when a fault of a specific functionality occurs. In particular, MSN reacts by instantiating a new instance of the functionality as an VNF via the MANO orchestrator. On the contrary, using a proactive approach for the orchestrator would guarantee less downtime service by leveraging intelligent algorithms to predict fault at functionalities. Second, we are considering the possibility to add intelligence at the orchestration level to dynamically and proactively manage network entities according to network behavior. For instance, if the network is not performing as expected (e.g., because of congestion), an intelligent orchestrator can predict that and can allocate useful functionalities to overcome the congestion. This vision can be extended to all system components including every single SDN (sub-)functionality, by bringing intelligence at functionality level. So, each functionality can set up its behavior to fit network requirements. Third, we are working on a resource provisioning strategy, where a plan is needed for the careful identification of network nodes, VNFs, and services that will fulfill the application requirements. In particular, the provisioning strategy must consider both applications and control network functionalities as resources to manage. Following the plan instructions, the orchestrator will deploy and configure all resources needed by the application.

Furthermore, we presented a multi-agent based grand architecture using MANA-NMS architecture. We developed a unified architecture combining SDN, NFV, MEC, and GANA architectural and conceptual models for future networks, such as 6G. The main principles addressed by the architecture are an agent-based design of network systems. Moreover, it also shows the organization of agents to produce an overall automated system. The architecture is expected to address future network demands in terms of flexibility, heterogeneity, reliability, and latency in an automated environment.

[1]For more details on the implementation and to reproduce the presented tests, we provide the entire project at `https://gitlab.com/dscotece/ryu_sdn_decomposition/`

Bibliography

[1] *3GPP TR 21.915 version 15.0.0 Release 15*. [Online; accessed 01-May-2021].

[2] 5G-PPP. *White Paper: 5G and the Factories of the Future*. URL: https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-White-Paper-on-Factories-of-the-Future-Vertical-Sector.pdf. (accessed: 01.11.2020).

[3] I. Afolabi et al. "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions". In: *IEEE Commu. Surveys Tutorials* 20.3 (2018.), 2429–2453.

[4] S. Agarwal et al. "VNF Placement and Resource Allocation for the Support of Vertical Services in 5G Networks". In: *IEEE/ACM Transactions on Networking* 27.1 (2019), pp. 433–446. ISSN: 1558-2566. DOI: 10.1109/TNET.2018.2890631.

[5] S. Ahvar et al. "SET: a Simple and Effective Technique to improve cost efficiency of VNF placement and chaining algorithms for network service provisioning". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. 2018, pp. 293–297. DOI: 10.1109/NETSOFT.2018.8459908.

[6] Zeinab Akhavan et al. "Internet of Things-enabled Passive Contact Tracing in Smart Cities". In: *Internet of Things* (2021), p. 100397.

[7] A. Gharaibeh et al. "Smart Cities: A Survey on Data Management, Security, and Enabling Technologies." In: *IEEE Communications Surveys & Tutorials* 19.4 (2017), pp. 2456–2501. DOI: 10.1109/COMST.2017.2736886.

[8] I. Al-Oqily and A. Karmouch. "A self-organizing composition towards autonomic overlay networks". In: *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*. Salvador, Bahia, Brazil, April 2008, pp. 287–294.

[9] O. Alhussein et al. "Joint VNF Placement and Multicast Traffic Routing in 5G Core Networks". In: *2018 IEEE Global Communications Conference (GLOBECOM)*. 2018, pp. 1–6. DOI: 10.1109/GLOCOM.2018.8648029.

[10] A. Alleg et al. "Delay-aware VNF placement and chaining based on a flexible resource allocation approach". In: *2017 13th International Conference on Network and Service Management (CNSM)*. 2017, pp. 1–7. DOI: 10.23919/CNSM.2017.8255993.

[11] D. Amaya et al. "VNF Placement with Optimization Problem Based on Data Throughput for Service Chaining". In: *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. 2018, pp. 1–3. DOI: 10.1109/CloudNet.2018.8549543.

[12] T. Amudha, N. Saritha, and P. Usha. "A Multi-agent based framework for Dynamic Service Discovery and access using matchmaking approach". In: *2009 International Conference on Intelligent Agent Multi-Agent Systems*. Chennai, India, July 2009, pp. 1–4.

[13] Osama Arouk and Navid Nikaein. "5G Cloud-Native: Network Management & Automation". In: *IEEE Symposium on Network Operations and Management* (2020).

[14] S. T. Arzo et al. "Multi-Agent Based Autonomic Network Management Architecture". In: *IEEE Transactions on Network and Service Management* (2021), pp. 1–1. DOI: 10.1109/TNSM.2021.3059752.

[15] S. T. Arzo et al. "Study of Virtual Network Function Placement in 5G Cloud Radio Access Network". In: *IEEE Transactions on Network and Service Management* (2020), pp. 1–1. ISSN: 1932-4537. DOI: 10.1109/TNSM.2020.3020390.

[16] Sisay Tadesse Arzo et al. "Multi-Agent Based Autonomic Network Management Architecture". In: *IEEE Transactions on Network and Service Management* (2021), pp. 1–1. DOI: 10.1109/TNSM.2021.3059752.

[17] G. Auer et al. "How much energy is needed to run a wireless network?" In: *IEEE Wireless Communications* 18.5 (2011), pp. 40–49. ISSN: 1536-1284. DOI: 10.1109/MWC.2011.6056691.

[18] S. Ayoubi, Y. Zhang, and C. Assi. "RAS:Reliable auto-scaling of virtual machines in multi-tenant cloud networks". In: *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*. 2015, pp. 1–6. DOI: 10.1109/CloudNet.2015.7335271.

[19] A. Babuscia, D. Divsalar, and K. Cheung. "CDMA communication system for mars areostationary relay satellite". In: *2017 IEEE Aerospace Conference*. 2017, pp. 1–10. DOI: 10.1109/AERO.2017.7943941.

[20] G. Baggio, R. Bassoli, and F. Granelli. "Cognitive Software-Defined Networking Using Fuzzy Cognitive Maps". In: *IEEE Transactions on Cognitive Communications and Networking* 5.3 (2019), pp. 517–539. DOI: 10.1109/TCCN.2019.2920593.

[21] F. Bannour, S. Souihi, and A. Mellouk. "Distributed SDN Control: Survey, Taxonomy, and Challenges". In: *IEEE Communications Surveys Tutorials* 20.1 (2018), pp. 333–354. DOI: 10.1109/COMST.2017.2782482.

[22] F. Bannour, S. Souihi, and A. Mellouk. "Survey, taxonomy, and challenges in IEEE Communications Surveys Tutorials". In: *IEEE Communications Surveys Tutorials* 20.1 (2018.), 333–354.

[23] A. A. Barakabitze et al. "5G network slicing using SDN and NFV". In: *A survey of taxonomy, architectures and future challenges in Computer Networks* 167 (2020.).

[24] F. Baroncelli, B. Martini, and P. Castoldi. "Autonomic service provisioning for Automatically Switched Transport Network". In: *COIN-NGNCON 2006 - The Joint International Conference on Optical Internet and Next Generation Network*. Jeju, South Korea, July 2006, pp. 15–17.

[25] R. Bassoli and F. Granelli. "An Algebraic Approach to Network Slicing". In: *European Wireless 2019; 25th European Wireless Conference*. 2019, pp. 1–6.

[26] R. Bassoli et al. "Towards 5G Cloud Radio Access Network: An Energy and Latency Perspective". In: *Transactions on Emerging Telecommunications Technologies* 27.1 (2019), pp. 433–446. ISSN: 1063-6692. DOI: 10.1002/ett.3669.

[27] Riccardo Bassoli. "Chapter 7 - Network function virtualization". In: *Computing in Communication Networks*. Ed. by Frank H.P. Fitzek, Fabrizio Granelli, and Patrick Seeling. Academic Press, 2020, pp. 119–132. ISBN: 978-0-12-820488-7.

[28] E. Bastug et al. “Toward Interconnected Virtual Reality: Opportunities, Challenges, and Enablers”. In: *IEEE Communications Magazine* 55.6 (2017), pp. 110–117. ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1601089.

[29] M. Behringer et al. “Autonomic Networking - from theory to practice”. In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. 2014, pp. 1–17.

[30] Michael Behringer, John Strassner, and Joel Halpern. *The Use of Control Loops in Autonomic Networking*. [Accessed:31-Aug-2020].

[31] O. Bello, S. Zeadally, and M. Badra. In: ().

[32] F. Ben Jemaa, G. Pujolle, and M. Pariente. “QoS-Aware VNF Placement Optimization in Edge-Central Carrier Cloud Architecture”. In: *2016 IEEE Global Communications Conference (GLOBECOM)*. 2016, pp. 1–7. DOI: 10.1109/GLOCOM.2016.7842188.

[33] M. Bencheikh Lehocine and M. Batouche. “Self-management in IP networks using autonomic computing”. In: *2014 International Conference and Workshop on the Network of the Future (NOF)*. 2014, pp. 1–3.

[34] I. Benkacem et al. “Optimal VNFs Placement in CDN Slicing Over Multi-Cloud Environment”. In: *IEEE Journal on Selected Areas in Communications* 36.3 (2018), pp. 616–627. ISSN: 0733-8716. DOI: 10.1109/JSAC.2018.2815441.

[35] S. Bera, S. Misra, and A. V. Vasilakos. “Software-Defined Networking for Internet of Things: A Survey”. In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 1994–2008. DOI: 10.1109/JIOT.2017.2746186.

[36] Carlos J. Bernardos et al. *European Vision for the 6G Network Ecosystem*. Jun, 2021. DOI: 10.5281/zenodo.5007671.

[37] W. Berrayana, H. Youssef, and G. Pujolle. “A generic cross-layer architecture for autonomic network management with network wide knowledge”. In: *2012 8th International Wireless Communications and Mobile Computing Conference (IWCMC)*. Limassol, Cyprus, Aug. 2012, pp. 82–87.

[38] A. Binsahaq, T. R. Sheltami, and K. Salah. “A Survey on Autonomic Provisioning and Management of QoS in SDN Networks”. In: *IEEE Access* 7 (May 2019), pp. 73384–73435.

[39] R. Birke et al. “Failure Analysis of Virtual and Physical Machines: Patterns, Causes and Characteristics”. In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 2014, pp. 1–12. DOI: 10.1109/DSN.2014.18.

[40] Luiz Bittencourt et al. “The Internet of Things, Fog and Cloud continuum: Integration and challenges”. In: *Internet of Things* 3-4 (2018), 134–155. ISSN: 2542-6605.

[41] Stefano Bonafini et al. “Virtual Baseband Unit Splitting Exploiting Small Satellite Platforms”. In: *2020 IEEE Aerospace Conference*. Mar. 1, 2020, pp. 1–11. published.

[42] G. Bouabene et al. “The autonomic network architecture (ANA)”. In: *IEEE Journal on Selected Areas in Communications* 28.1 (December 2009), pp. 4–14.

[43] Raouf Boutaba et al. "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities". In: *Journal of Internet Services and Applications* 9.1 (2018), p. 16. ISSN: 1869-0238. DOI: 10.1186/s13174-018-0087-2. URL: https://doi.org/10.1186/s13174-018-0087-2.

[44] T. Braga et al. "A Tiny and Light-Weight Autonomic Element for Wireless Sensor Networks". In: *Fourth International Conference on Autonomic Computing (ICAC'07)*. Jacksonville, FL, USA, June 2007, pp. 17–17.

[45] M. A. B. Brasil et al. "Performance Comparison of Multi-Agent Middleware Platforms for Wireless Sensor Networks". In: *IEEE Sensors Journal* 18.7 (January 2018), pp. 3039–3049.

[46] J. Budakoti, A. S. Gaur, and C. Lung. "IoT Gateway Middleware for SDN Managed IoT". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, pp. 154–161.

[47] Calico. *Calico*. URL: https://www.projectcalico.org/. (accessed: 6.1.2021).

[48] R. Chai et al. "Multi-Objective Optimization-Based Virtual Network Embedding Algorithm for Software-Defined Networking". In: *IEEE Transactions on Network and Service Management* (2019), pp. 1–1. ISSN: 2373-7379. DOI: 10.1109/TNSM.2019.2953297.

[49] C. L. Chamas, D. Cordeiro, and M. M. Eler. "Comparing REST, SOAP, Socket and gRPC in computation offloading of mobile applications: An energy cost analysis." In: *IEEE 9th Latin-American Conference on Communications (LATINCOM)* (2017), pp. 1–6. DOI: 10.1109/LATINCOM.2017.8240185..

[50] R. Chaparadza et al. "Implementation Guide for the ETSI AFI GANA model: A Standardized Reference Model for Autonomic Networking, Cognitive Networking and Self-Management". In: *2013 IEEE Globecom Workshops (GC Wkshps)*. 2013, pp. 935–940.

[51] R. Chaparadza et al. "SDN enablers in the ETSI AFI GANA Reference Model for Autonomic Management Control (emerging standard), and Virtualization impact". In: *2013 IEEE Globecom Workshops (GC Wkshps)*. Atlanta, GA, USA, Dec. 2013, pp. 818–823.

[52] R. Chaparadza et al. "Standardization of resilience survivability, and autonomic fault-management, in evolving and future networks: An ongoing initiative recently launched in ETSI". In: *2013 9th International Conference on the Design of Reliable Communication Networks (DRCN)*. 2013, pp. 331–341.

[53] J. Chen et al. "On wireless sensor network mobile agent multi-objective optimization route planning algorithm". In: *2017 IEEE International Conference on Agents (ICA)*. Beijing, China, July 2017, pp. 101–103.

[54] X. Chen et al. "On Incentive-Driven VNF Service Chaining in Inter-Datacenter Elastic Optical Networks: A Hierarchical Game-Theoretic Mechanism". In: *IEEE Transactions on Network and Service Management* 16.1 (2019), pp. 1–12. ISSN: 2373-7379. DOI: 10.1109/TNSM.2018.2866400.

[55] Z. Cheng, T. Wang, and Y. Xin. "High-Order Distributed Consensus in Multi-Agent Networks". In: *2018 IEEE 7th Data Driven Control and Learning Systems Conference (DDCLS)*. Enshi, China, May 2018, pp. 965–969.

[56] Margaret BT Chiosi et al. *Network Functions Virtualisation*. Tech. rep. URL: http://portal.etsi.org/NFV/NFV{_}White{_}Paper.pdf.

[57] F. Chiti et al. "Virtual Functions Placement With Time Constraints in Fog Computing: A Matching Theory Perspective". In: *IEEE Transactions on Network and Service Management* 16.3 (2019), pp. 980–989. ISSN: 2373-7379. DOI: 10.1109/TNSM.2019.2918637.

[58] D. Cho et al. "Real-Time Virtual Network Function (VNF) Migration toward Low Network Latency in Cloud Environments". In: *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. 2017, pp. 798–801. DOI: 10.1109/CLOUD.2017.118.

[59] M. Chowdhury, M. R. Rahman, and R. Boutaba. "ViNEYard: Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping". In: *IEEE/ACM Transactions on Networking* 20.1 (2012), pp. 206–219. ISSN: 1063-6692. DOI: 10.1109/TNET.2011.2159308.

[60] CNCF. *Kubernetes*. URL: https://kubernetes.io/. (accessed: 14.12.2020).

[61] D. Comer and A. Rastegarnia. "Externalization of Packet Processing in Software Defined Networking". In: *IEEE Networking Letters* 1.3 (2019), pp. 124–127. ISSN: 2576-3156. DOI: 10.1109/LNET.2019.2918155.

[62] Douglas Comer and Adib Rastegarnia. "Towards Disaggregating the SDN Control Plane". In: *CoRR* abs/1902.00581 (2019). arXiv: 1902.00581. URL: http://arxiv.org/abs/1902.00581.

[63] *Consensus on 6G is Gradually Forming*. Accessed: 2020-03-24.

[64] *CONTAINERS ON VIRTUAL MACHINES OR BARE METAL?*

[65] D. Cotroneo et al. "Network Function Virtualization: Challenges and Directions for Reliability Assurance". In: *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. 2014, pp. 37–42. DOI: 10.1109/ISSREW.2014.48.

[66] G. Cui et al. "Latency and Energy Optimization for MEC Enhanced SAT-IoT Networks". In: *IEEE Access* 8 (2020), pp. 55915–55926. DOI: 10.1109/ACCESS.2020.2982356.

[67] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros. "Dynamic Latency-Optimal VNF Placement at Network Edge". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 693–701. DOI: 10.1109/INFOCOM.2018.8486021.

[68] A. Daga et al. "Terrain-based simulation of IEEE 802.11a and b physical layers on the martian surface". In: *IEEE Transactions on Aerospace and Electronic Systems* 43.4 (2007), pp. 1617–1624. DOI: 10.1109/TAES.2007.4441762.

[69] E. Dahlman et al. "5G wireless access: requirements and realization". In: *IEEE Communications Magazine* 52.12 (2014), pp. 42–47. ISSN: 0163-6804. DOI: 10.1109/MCOM.2014.6979985.

[70] T. Das and M. Gurusamy. "Controller Placement for Resilient Network State Synchronization in Multi-Controller SDN." In: *IEEE Communications Letters* 24.6 (2020), pp. 1299–1303. DOI: 10.1109/LCOMM.2020.2979072..

[71] E. M. Davidson and S. D. J. McArthur. "Exploiting Multi-agent System Technology within an Autonomous Regional Active Network Management System". In: *2007 International Conference on Intelligent Systems Applications to Power Systems*. Toki Messe, Niigata, Japan, Nov. 2007, pp. 1–6.

[72] *DEDICAT 6G: Dynamic coverage Extension and Distributed Intelligence for human Centric Applications with assured security, pri- vacy, and Trust: from 5G to 6G*. Accessed: 19-Oct-2021. 2021.

[73] *Deep Space Network*. Accessed: 26-Feb-2021.

[74] *Deliverable D03.01 Report on IoT platform activities - UNIFY-IoT*.

[75] B. Denby and B. Lucia. "Orbital Edge Computing: Machine Inference in Space". In: *IEEE Computer Architecture Letters* 18.1 (2019), pp. 59–62. DOI: 10.1109/LCA.2019.2907539.

[76] H. Derhamy, J. Eliasson, and J. Delsing. "IoT Interoperability—On-Demand and Low Latency Transparent Multiprotocol Translator". In: *IEEE Internet of Things Journal* 4.5 (2017), pp. 1754–1763.

[77] M. Dieye et al. "CPVNF: Cost-Efficient Proactive VNF Placement and Chaining for Value-Added Services in Content Delivery Networks". In: *IEEE Transactions on Network and Service Management* 15.2 (2018), pp. 774–786. ISSN: 1932-4537. DOI: 10.1109/TNSM.2018.2815986.

[78] Docker docs. *Install Docker Engine on Ubuntu*. URL: https://docs.docker.com/engine/install/ubuntu/. (accessed: 26.12.2020).

[79] K. Dolui and S. K. Datta. "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing". In: *2017 Global Internet of Things Summit (GIoTS)*. Geneva, Switzerland, June 2017, pp. 1–6.

[80] A. Dorri, S. S. Kanhere, and R. Jurdak. "Multi-Agent Systems: A Survey". In: *IEEE Access* 6 (April 2018), pp. 28573–28593.

[81] M. L. Dowell and R. D. Bonnell. "Learning for distributed artificial intelligence systems". In: *[1991 Proceedings] The Twenty-Third Southeastern Symposium on System Theory*. Columbia, SC, USA, Mar. 1991, pp. 218–221.

[82] M. ElGammal and M. Eltoweissy. "Towards Aware, Adaptive and Autonomic Sensor-Actuator Networks". In: *2011 IEEE Fifth International Conference on Self-Adaptive and Self-Organizing Systems*. 2011, pp. 210–211.

[83] H. ElSawy et al. "Modeling and Analysis of Cellular Networks Using Stochastic Geometry: A Tutorial". In: *IEEE Communications Surveys Tutorials* 19.1 (2017), pp. 167–203. ISSN: 2373-745X. DOI: 10.1109/COMST.2016.2624939.

[84] A. Engelmann and A. Jukan. "A Reliability Study of Parallelized VNF Chaining". In: *2018 IEEE International Conference on Communications (ICC)*. 2018, pp. 1–6. DOI: 10.1109/ICC.2018.8422595.

[85] " ETSI". "Next Generation Protocols (NGP)- Intelligence Defined Network (IDN) ". In: (2018).

[86] "ETSI". "Autonomic network engineering for the self-managing Future Internet (AFI): Scenarios, Use Cases and Requirements for Autonomic/Self-Managing Future Internet". In: (2011).

[87] "ETSI". "Network Technologies (NTECH): Autonomic network engineering for the self-managing Future Internet (AFI): Autonomicity and Self-Management in the Broadband Forum (BBF) Architectures". In: (2018).

[88] Fangxin Wang et al. "Bandwidth guaranteed virtual network function placement and scaling in datacenter networks". In: *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*. 2015, pp. 1–8. DOI: 10.1109/PCCC.2015.7410276.

[89] P. Farkaš. "Adaptive Feedback Supported Communication for IoT and Space Applications". In: *2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2019, pp. 61–64. DOI: 10.1109/AHS.2019.00005.

[90] P. V. R. Ferreira et al. "Multi-objective reinforcement learning-based deep neural networks for cognitive space communications". In: *2017 Cognitive Communications for Aerospace Applications Workshop (CCAA)*. 2017, pp. 1–8. DOI: 10.1109/CCAAW.2017.8001880.

[91] C. Fiandrino et al. "Performance and Energy Efficiency Metrics for Communication Systems of Cloud Computing Data Centers". In: *IEEE Transactions on Cloud Computing* 5.4 (2017), pp. 738–750. ISSN: 2168-7161. DOI: 10.1109/TCC.2015.2424892.

[92] Frank H. P. Fitzek, Fabrizio Granelli, and Patrick Seeling, eds. *Computing in Communication Networks - From Theory to Practice*. 1st ed. Vol. 1. 1. Elsevier, Jan. 1, 2020. ISBN: tbd. published.

[93] *Floodlight: Floodlight Controller*. 2021.

[94] *Flow Monitor*.

[95] Giancarlo Fortino et al. "Towards Multi-layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach". en. In: *Integration, Interconnection, and Interoperability of IoT Systems*. Ed. by Raffaele Gravina et al. Internet of Things. Cham: Springer International Publishing, 2018, pp. 199–232. ISBN: 978-3-319-61300-0. DOI: 10.1007/978-3-319-61300-0_10. (Visited on 08/17/2020).

[96] Open Networking Foundation. *Open Network Operating System (ONOS)*. URL: https://docs.onosproject.org/.

[97] "Open Networking Foundation". "Software-Defined Networking: The New Norm for Networks". In: (2012).

[98] A. Galis, S. Clayman, and L. Mamatas. "Towards autonomic management of software enabled networks". In: *2013 8TH International Symposium on Advanced Topics in Electrical Engineering (ATEE)*. Bucharest, Romania, May 2013, pp. 1–3.

[99] G. Garg et al. "DAVIS: A Delay-Aware VNF Selection Algorithm for Service Function Chaining". In: *2019 11th International Conference on Communication Systems Networks (COMSNETS)*. 2019, pp. 436–439. DOI: 10.1109/COMSNETS.2019.8711442.

[100] Sahil Garg et al. "SDN-NFV-Aided Edge-Cloud Interplay for 5G-Envisioned Energy Internet Ecosystem". In: *IEEE Network* 35.1 (2021), pp. 356–364. DOI: 10.1109/MNET.011.1900602.

[101] D. Gavalas et al. "A hybrid centralised-distributed network management architecture". In: *Proceedings IEEE International Symposium on Computers and Communications (Cat. No.PR00250)*. 1999, pp. 434–441. DOI: 10.1109/ISCC.1999.780940.

[102] Michael R. Genesereth and Steven P. Ketchpel. "Software Agents". In: *Commun. ACM* 37.7 (July 1994), 48–ff. ISSN: 0001-0782. DOI: 10.1145/176789.176794. URL: https://doi.org/10.1145/176789.176794.

[103] M. Ghaznavi et al. In: *IEEE Journal on Selected Areas in Communications, title=Distributed Service Function Chaining* 35.11 (2017), pp. 2479–2489. ISSN: 1558-0008. DOI: 10.1109/JSAC.2017.2760178.

[104] H. Gogineni et al. "MMS: An autonomic network-layer foundation for network management". In: *IEEE Journal on Selected Areas in Communications* 28.1 (2010), pp. 15–27.

[105] F. Granelli and R. Bassoli. "Autonomic Mobile Virtual Network Operators for Future Generation Networks". In: *IEEE Network* 32.5 (September 2018), pp. 76–84. DOI: 10.1109/MNET.2018.1700455.

[106] F. Granelli and R. Bassoli. "Towards Autonomic Mobile Network Operators". In: *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. 2018, pp. 1–4. DOI: 10.1109/CloudNet.2018.8549552.

[107] Fabrizio Granelli, Dzmitry Kliazovich, and Neumar" Malheiros. "Towards Cognitive Internet: An Evolutionary Vision". In: *Cognitive Radio and Networking for Heterogeneous Wireless Networks: Recent Advances and Visions for the Future* (2015), pp. 181–200. DOI: 10.1007/978-3-319-01718-1_6.

[108] Aditya Gudipati et al. "SoftRAN: software defined radio access network." In: 2018. 25–30. DOI: 10.1145/2491185.2491207.

[109] R. Guerzoni et al. "Modeling Reliability Requirements in Coordinated Node and Link Mapping". In: *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*. 2014, pp. 321–330. DOI: 10.1109/SRDS.2014.17.

[110] M. Haenggi et al. "Stochastic geometry and random graphs for the analysis and design of wireless networks". In: *IEEE Journal on Selected Areas in Communications* 27.7 (2009), pp. 1029–1046. ISSN: 1558-0008. DOI: 10.1109/JSAC.2009.090902.

[111] O. Hahm et al. "Operating Systems for Low-End Devices in the Internet of Things: A Survey". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 720–734. DOI: 10.1109/JIOT.2015.2505901.

[112] Aboul Ella Hassanien, Ashraf Darwish, and Sara Abdelghafar. "Machine learning in telemetry data mining of space mission: basics, challenging and future directions". In: *Artificial Intelligence Review* 53 (2020), pp. 1573–7462. DOI: 10.1007/s10462-019-09760-1. URL: https://doi.org/10.1007/s10462-019-09760-1.

[113] H. Hawilo, M. Jammal, and A. Shami. "Network Function Virtualization-Aware Orchestrator for Service Function Chaining Placement in the Cloud". In: *IEEE Journal on Selected Areas in Communications* 37.3 (2019), pp. 643–655. ISSN: 0733-8716. DOI: 10.1109/JSAC.2019.2895226.

[114] H. Hawilo, M. Jammal, and A. Shami. "Network Function Virtualization-Aware Orchestrator for Service Function Chaining Placement in the Cloud". In: *IEEE Journal on Selected Areas in Communications* 37.3 (2019), pp. 643–655. ISSN: 1558-0008. DOI: 10.1109/JSAC.2019.2895226.

[115] Hexa-X. *D1.2 – Expanded 6G vision, use cases and societal values — including aspects of sustainability, security and spectrum*. 2021. URL: https://hexa-x.eu/wp-content/uploads/2021/05/Hexa-X_D1.2.pdf (visited on 2021).

[116] Shiku Hirai et al. "Automated Provisioning of Cloud-Native Network Functions in Multi-Cloud Environments". In: *IEEE Conference on Network Softwarization (NetSoft)* (2020), pp. 359–361.

[117] A. Hirwe and K. Kataoka. "LightChain: A lightweight optimisation of VNF placement for service chaining in NFV". In: *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. 2016, pp. 33–37. DOI: 10.1109/NETSOFT.2016.7502438.

[118] et al. Hong. *IoT Edge Challenges and Functions draft-irtf-t2trg-iot-edge-00*. Accessed: 20-Feb-2021. Expires 4 March 2021.

[119] *How Big is IoT? 20.6 Billion Connected Devices By 2020.* Accessed: Dec. 03, 2020. URL: {https://mitechnews.com/internet-of-things/how-big-is-iot-20-6-billion-connected-devices-by-2020/\#:~:text=Big\%20is\%20IoT\%3F-,20.6\%20Billion\%20Connected\%20Devices\%20By\%202020.,smart\%20cities\%20and\%20connected\%20healthcare.}.

[120] K. Huang, S. Srivastava, and D. Cartes. "Decentralized Reconfiguration for Power Systems Using Multi Agent System". In: *2007 1st Annual IEEE Systems Conference*. Honolulu, HI, USA, April 2007, pp. 1–6.

[121] S. Hussain, E. Shakshuki, and A. W. Matin. "Agent-based system architecture for wireless sensor networks". In: *20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*. Vol. 2. Vienna, Austria, April 2006, pp. 1–5.

[122] N. Hyodo et al. "Virtual Network Function Placement Model for Service Chaining to Relax Visit Order and Routing Constraints". In: *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. 2018, pp. 1–3. DOI: 10.1109/CloudNet.2018.8549553.

[123] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008). 2020.

[124] *Is the IoT in space about to take off?* https://www.networkworld.com/article/3315736/is-the-iot-in-space-about-to-take-off.html.

[125] M. Ismail et al. "A Survey on Green Mobile Networking: From The Perspectives of Network Operators and Mobile Users". In: *IEEE Communications Surveys Tutorials* 17.3 (2015), pp. 1535–1556. ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2367592.

[126] M. Jaber et al. "5G Backhaul Challenges and Emerging Research Directions: A Survey". In: *IEEE Access* 4 (2016), pp. 1743–1766. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2556011.

[127] Jae Chul Moon and Soon Ju Kang. "A multi-agent architecture for intelligent home network service using tuple space model". In: *IEEE Transactions on Consumer Electronics* 46.3 (Aug 2000), pp. 791–794.

[128] Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. "A Roadmap of Agent Research and Development". In: *Autonomous Agents and Multi-Agent Systems* 1.1 (Jan. 1998), 7–38. ISSN: 1387-2532. DOI: 10.1023/A:1010090405266. URL: https://doi.org/10.1023/A:1010090405266.

[129] Sprague Jhon. *NASA's Internet of Things Lab*. https://internet-of-things.cioreview.com/cxoinsight/nasa-s-internet-of-things-lab--nid-13174-cid-133.html.

[130] Q. Jiang et al. "Research on Load Balancing Based on Multi-agent in Ubiquitous Networks". In: *2010 International Conference on Intelligent Computation Technology and Automation*. Vol. 3. Changsha, China, May 2010, pp. 10–13.

[131] B. Kar, E. H. Wu, and Y. Lin. "Energy Cost Optimization in Dynamic Placement of Virtualized Network Function Chains". In: *IEEE Transactions on Network and Service Management* 15.1 (2018), pp. 372–386. ISSN: 1932-4537. DOI: 10.1109/TNSM.2017.2782370.

[132] M. Karimzadeh-Farshbafan, V. Shah-Mansour, and D. Niyato. “Reliability Aware Service Placement Using a Viterbi-based Algorithm”. In: *IEEE Transactions on Network and Service Management* (2019), pp. 1–1. ISSN: 2373-7379. DOI: 10.1109/TNSM.2019.2959818.

[133] J. O. Kephart and D. M. Chess. “The vision of autonomic computing”. In: *Computer* 36.1 (January 2003), pp. 41–50. ISSN: 1558-0814. DOI: 10.1109/MC.2003.1160055.

[134] S. Khebbache, M. Hadji, and D. Zeghlache. “A multi-objective non-dominated sorting genetic algorithm for VNF chains placement”. In: *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 2018, pp. 1–4. DOI: 10.1109/CCNC.2018.8319250.

[135] H. Kim and N. Feamster. “Improving network management with software defined networking”. In: *IEEE Commun. Mag.* 51.2 (2013), 114–119.

[136] Dzmitry Kliazovich et al. “e-STAB: Energy-Efficient Scheduling for Cloud Computing Applications with Traffic Load Balancing”. In: *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. 2013, pp. 7–13. DOI: 10.1109/GreenCom-iThings-CPSCom.2013.28.

[137] K. Kondepu et al. “Experimental Demonstration of 5G Virtual EPC Recovery in Federated Testbeds”. In: *IFIP/IEEE International Symposium on Integrated Network Management* (2020), pp. 712–713.

[138] P. Kookarinrat and Y. Temtanapat. “Design and implementation of a decentralized message bus for microservices”. In: *13th International Joint Conference on Computer Science and Software Engineering (JCSSE)* (2016), pp. 1–6. DOI: doi:10.1109/JCSSE.2016.7748869..

[139] Alexander Kott et al. *Autonomous Intelligent Cyber-defense Agent (AICA) Reference Architecture. Release 2.0*. 2018. arXiv: 1803.10664 [cs.CR].

[140] Herleen Kour and Naveen Gondhi. “Machine Learning Techniques: A Survey”. In: *Innovative Data Communication Technologies and Application*. Ed. by Jennifer S. Raj, Abul Bashar, and S. R. Jino Ramson. Cham: Springer International Publishing, 2020, pp. 266–275. ISBN: 978-3-030-38040-3.

[141] Y. Koyasako et al. “Real-Time Motion Control Method Using Measured Delay Information on Access Edge Computing”. In: *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*. 2020, pp. 1–4. DOI: 10.1109/CCNC46108.2020.9045470.

[142] A. Laghrissi et al. “Towards Edge Slicing: VNF Placement Algorithms for a Dynamic amp; Realistic Edge Cloud Environment”. In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. 2017, pp. 1–6. DOI: 10.1109/GLOCOM.2017.8254653.

[143] J. M. Landsberg. *Tensors: Geometry and Applications*. American Mathematical Society, 2012, pp. 1–493.

[144] S. Lange et al. “A Multi-objective Heuristic for the Optimization of Virtual Network Function Chain Placement”. In: *2017 29th International Teletraffic Congress (ITC 29)*. Vol. 1. 2017, pp. 152–160. DOI: 10.23919/ITC.2017.8064351.

[145] David Lary. “Artificial Intelligence in Aerospace”. In: Jan. 2010. ISBN: 978-953-7619-96-1. DOI: 10.5772/6941.

[146] Alexandru Lavric and Valentin Popa. *Performance Evaluation of LoRaWAN Communication Scalability in Large-Scale Wireless Sensor Networks*. en. Research Article. ISSN: 1530-8669 Publisher: Hindawi Volume: 2018. June 2018. DOI: `https://doi.org/10.1155/2018/6730719`.

[147] W. Lee, T. Na, and J. Kim. "How to Create a Network Slice? - A 5G Core Network Perspective". In: *2019 21st International Conference on Advanced Communication Technology (ICACT)*. PyeongChang Kwangwoon_Do, Korea, Feb. 2019, pp. 616–619.

[148] Ángel Leonardo Valdivieso Caraguay. et al. "SDN/NFV Architecture for IoT Networks". In: *Proceedings of the 14th International Conference on Web Information Systems and Technologies - Volume 1: ITSCO*, INSTICC. SciTePress, 2018, pp. 425–429. ISBN: 978-989-758-324-7. DOI: `10.5220/0007234804250429`.

[149] Kristina Lerman, Aram Galstyan, and Tad Hogg. *Mathematical Analysis of Multi-Agent Systems*. 2004. arXiv: `cs/0404002 [cs.RO]`.

[150] D. Li et al. "Availability Aware VNF Deployment in Datacenter Through Shared Redundancy and Multi-Tenancy". In: *IEEE Transactions on Network and Service Management* 16.4 (2019), pp. 1651–1664. ISSN: 2373-7379. DOI: `10.1109/TNSM.2019.2936505`.

[151] D. Li et al. "Virtual Network Function Placement Considering Resource Optimization and SFC Requests in Cloud Datacenter". In: *IEEE Transactions on Parallel and Distributed Systems* 29.7 (2018), pp. 1664–1677. ISSN: 1045-9219. DOI: `10.1109/TPDS.2018.2802518`.

[152] H. Li et al. "An autonomic management architecture for SDN-based multi-service network". In: *2013 IEEE Globecom Workshops (GC Wkshps)*. 2013, pp. 830–835.

[153] Z. Li and Y. Yang. "Placement of Virtual Network Functions in Hybrid Data Center Networks". In: *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*. 2017, pp. 73–79. DOI: `10.1109/HOTI.2017.15`.

[154] C. Liao et al. "Wireless Sensor Network Performance Research for Leach Based on Multi-agent Simulation". In: *2016 IEEE International Conference on Agents (ICA)*. Matsue, Japan, Sept. 2016, pp. 98–99.

[155] G. Liu and D. Juan. "Design of Distributed Network Management System Based on Multi-agent". In: *2010 Third International Symposium on Information Processing*. Qingdao, China, Oct. 2010, pp. 433–436.

[156] X. Liu, P. Juluri, and D. Medhi. "An experimental study on dynamic network reconfiguration in a virtualized network environment using autonomic management". In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. 2013, pp. 616–622.

[157] B. Loganayagi and S. Sujatha. "Creating virtual platform for cloud computing". In: *2010 IEEE International Conference on Computational Intelligence and Computing Research*. Coimbatore, India, Dec. 2010, pp. 1–4.

[158] D. Lohani, P. Singh, and S. Varma. "Multi-Agent data aggregation in wireless sensor network using source grouping". In: *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. New Delhi, India, Sept. 2014, pp. 2174–2179.

[159] X. Long et al. "Autonomic Networking: Architecture Design and Standardization". In: *IEEE Internet Computing* 21.5 (2017), pp. 48–53.

[160] A. A. F. Loureiro and L. B. Ruiz. “Autonomic Wireless Networks in Smart Environments”. In: *Fifth Annual Conference on Communication Networks and Services Research (CNSR ’07)*. 2007, pp. 5–7.

[161] Wei Lu and Marco Di Renzo. “Stochastic Geometry Modeling of Cellular Networks: Analysis, Simulation and Experimental Validation”. In: *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. MSWiM ’15. Cancun, Mexico: ACM, 2015, pp. 179–188. ISBN: 978-1-4503-3762-5. DOI: 10.1145/2811587.2811597. URL: http://doi.acm.org/10.1145/2811587.2811597.

[162] M. Y. Lyazidi, N. Aitsaadi, and R. Langar. “Resource Allocation and Admission Control in OFDMA-Based Cloud-RAN”. In: *2016 IEEE Global Communications Conference (GLOBECOM)*. 2016, pp. 1–6. DOI: 10.1109/GLOCOM.2016.7842217.

[163] Martin Lévesque and David Tipper. “A Survey of Clock Synchronization Over Packet-Switched Networks”. In: *IEEE Communications Surveys & Tutorials* 18.4 (2016), pp. 2926–2947. DOI: 10.1109/COMST.2016.2590438.

[164] M. Guizani K. Shuaib M. A. Salahuddin A. Al-Fuqaha and F. Sallabi. “Softwarization of Internet of Things Infrastructure for Secure and Smart Healthcare in Computer.” In: *Journal of Cleaner Production* 50.7 (2017), pp. 74–79. DOI: 10.1109/MC.2017.195..

[165] S. Rommer L. Frid M. Olsson S. Sultana and C. Mulligan. *SAE and the Evolved Packet Core: Driving the Mobile Broadband Revolution.* Accessed: 19-Oct-2021. 2021.

[166] C. Ma et al. “ABSR: An Agent Based Self-Recovery Model for Wireless Sensor Network”. In: *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*. Chengdu, China, Dec. 2009, pp. 400–404.

[167] Y. Ma et al. “Exploring Edge Computing for Multitier Industrial Control”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pp. 3506–3518. DOI: 10.1109/TCAD.2020.3012648.

[168] M. Maier et al. “The tactile internet: vision, recent progress, and open challenges”. In: *IEEE Communications Magazine* 54.5 (2016), pp. 138–145. DOI: https://doi.org/10.1109/MCOM.2016.7470948..

[169] G. Marchetto et al. “Formally verified latency-aware VNF placement in industrial Internet of things”. In: *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. 2018, pp. 1–9. DOI: 10.1109/WFCS.2018.8402355.

[170] B. Mathieu et al. “Autonomic Management of Context-Aware Ambient Overlay Networks”. In: *2007 Second International Conference on Communications and Networking in China*. 2007, pp. 727–733.

[171] M. May et al. “Monitoring as first class citizen in an autonomic network universe”. In: *2007 2nd Bio-Inspired Models of Network, Information and Computing Systems*. Budapest, Hungary, Dec. 2007, pp. 247–254. DOI: 10.1109/BIMNICS.2007.4610121.

[172] J. A. McCann and R. Sterritt. “Autonomic Pervasive Networks (APNs) - Extended Abstract”. In: *2010 Seventh IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems*. 2010, pp. 145–148.

[173] N. McKeown et al. “Openflow: Enabling innovation in campus networks in SIGCOMM Comput.Commun. Rev.” In: 38.2 (2008), 69–74.

[174] M. Mechtri, C. Ghribi, and D. Zeghlache. “A Scalable Algorithm for the Placement of Service Function Chains”. In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 533–546. ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2598068.

[175] M. Mechtri, C. Ghribi, and D. Zeghlache. “VNF Placement and Chaining in Distributed Cloud”. In: *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. 2016, pp. 376–383. DOI: 10.1109/CLOUD.2016.0057.

[176] Anu Mercian et al. “UDAAN: Embedding User-Defined Analytics Applications in Network Devices”. In: *Proceedings of the 2019 Workshop on Network Meets AI & ML*. NetAI’19. Beijing, China: Association for Computing Machinery, 2019, 70–75. ISBN: 9781450368728. DOI: 10.1145/3341216.3342216. URL: https://doi.org/10.1145/3341216.3342216.

[177] "Tayeb Ben Meriem et al. “GANA - Generic Autonomic Networking Architecture”. In: (2016).

[178] Justin Meza et al. “A Large Scale Study of Data Center Network Reliability”. In: *Proceedings of the Internet Measurement Conference 2018*. IMC ’18. Boston, MA, USA: ACM, 2018, pp. 393–407. ISBN: 978-1-4503-5619-0. DOI: 10.1145/3278532.3278566. URL: http://doi.acm.org/10.1145/3278532.3278566.

[179] *Mobile Edge Computing A key technology towards 5G - White Paper*. https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf. [Online; accessed 01-May-2021].

[180] M. Mohammadian. “Network Security Risk Assessment Using Intelligent Agents”. In: *2018 International Symposium on Agent, Multi-Agent Systems and Robotics (ISAMSR)*. Putrajaya, Malaysia, Aug. 2018, pp. 1–6.

[181] O. Mohammed and J. Kianfar. “A Machine Learning Approach to Short-Term Traffic Flow Prediction: A Case Study of Interstate 64 in Missouri”. In: *2018 IEEE International Smart Cities Conference (ISC2)*. 2018, pp. 1–7. DOI: 10.1109/ISC2.2018.8656924.

[182] B. Mokhtar and M. Eltoweissy. “Memory-enabled autonomic resilient networking”. In: *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*. 2011, pp. 132–141.

[183] Benjamin Molina. *INTER-Iot - Interoperability Internet of Things*. en-US. (Visited on 08/18/2020).

[184] C. Mouradian, N. T. Jahromi, and R. H. Glitho. “NFV and SDN-Based Distributed IoT Gateway for Large-Scale Disaster Management”. In: *IEEE Internet of Things Journal* 5.5 (2018), pp. 4119–4131. DOI: 10.1109/JIOT.2018.2867255.

[185] C. Mouradian, S. Kianpisheh, and R. H. Glitho. “Application Component Placement in NFV-based Hybrid Cloud/Fog Systems”. In: *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. 2018, pp. 25–30. DOI: 10.1109/LANMAN.2018.8475055.

[186] A. Moursy et al. “Testbed implementation for Autonomic Network Performance Management of wireless mesh networks”. In: *2012 IEEE Globecom Workshops*. 2012, pp. 903–907.

[187] Z. Movahedi et al. “A Survey of Autonomic Network Architectures and Evaluation Criteria”. In: *IEEE Communications Surveys Tutorials* 14.2 (May 2011), pp. 464–490.

[188] *Next G Alliance: Building the Foundation for North American Leadership in 6G and Beyond*. Accessed: 19-Oct-2021. 2021.

[189] *NFV Technology*. Accessed: 2019-10-30.

[190] D. T. Nguyen et al. “Placement and Chaining for Run-time IoT Service Deployment in Edge-Cloud”. In: *IEEE Transactions on Network and Service Management* (2019), pp. 1–1. ISSN: 2373-7379. DOI: 10.1109/TNSM.2019.2948137.

[191] J. C. Nobre and L. Z. Granville. “Towards Consistency of Policy States in Decentralized Autonomic Network Management”. In: *2009 IEEE International Symposium on Policies for Distributed Systems and Networks*. London, UK, July 2009, pp. 170–173.

[192] L. Noirie et al. “Semantic networking: Flow-based, traffic-aware, and self-managed networking”. In: *Bell Labs Technical Journal* 14.2 (2009), pp. 23–38.

[193] Nokia. *Nokia white paper on 5G for Mission Critical Communication*. Nokia, Finland, 2018.

[194] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. “Interoperability in Internet of Things Infrastructure: Classification, Challenges, and Future Work: Third International Conference, IoTaaS 2017, Taichung, Taiwan, September 20–22, 2017, Proceedings”. In: Jan. 2018, pp. 11–18. ISBN: 978-3-030-00409-5. DOI: 10.1007/978-3-030-00410-1_2.

[195] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. “Interoperability in Internet of Things Infrastructure: Classification, Challenges, and Future Work: Third International Conference, IoTaaS 2017, Taichung, Taiwan, September 20–22, 2017, Proceedings”. In: Jan. 2018, pp. 11–18. ISBN: 978-3-030-00409-5. DOI: 10.1007/978-3-030-00410-1_2.

[196] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. “Interoperability in Internet of Things: Taxonomies and Open Challenges”. In: *Mobile Networks and Applications* 24.3 (2019), pp. 796–809. ISSN: 1572-8153. DOI: 10.1007/s11036-018-1089-9.

[197] *NOX Controller*. Accessed: 2020-07-08.

[198] O. Obulesu, M. Mahendra, and M. ThrilokReddy. “Machine Learning Techniques and Tools: A Survey”. In: *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. Coimbatore, India, 2018, pp. 605–611. DOI: 10.1109/ICIRCA.2018.8597302.

[199] Yustus Eko Oktian et al. “A survey on design choice, Computer Networks.” In: *Journal of Cleaner Production* 121 (2017), pp. 100–111. ISSN: 1389-1286. DOI: 10.1109/MC.2017.195..

[200] J. Okwuibe et al. “SDN Enhanced Resource Orchestration of Containerized Edge Applications for Industrial IoT.” In: *IEEE Transactions on Network and Service Management* 8 (2020.), pp. 229117–229131. DOI: 10.1109/ACCESS.2020.3045563.

[201] R. Olfati-Saber, J. A. Fax, and R. M. Murray. “Consensus and Cooperation in Networked Multi-Agent Systems”. In: *Proceedings of the IEEE* 95.1 (March 2007), pp. 215–233.

[202] D. B. Oljira et al. “A model for QoS-aware VNF placement and provisioning”. In: *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2017, pp. 1–7. DOI: 10.1109/NFV-SDN.2017.8169829.

[203] ONOS. *micro ONOS*. [Online; accessed 01-May-2021]. 2020.

[204] *ONOS Controller*. Accessed: 2020-07-08.

[205] G. A. Oparin et al. “Distributed solvers of applied problems based on microservices and agent networks”. In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2018, pp. 1415–1420. DOI: 10.23919/MIPRO.2018.8400255.

[206] *OpenDaylight Controller*. Accessed: 08-Jul-2020.

[207] M. Otokura et al. “Evolvable Virtual Network Function Placement Method: Mechanism and Performance Evaluation”. In: *IEEE Transactions on Network and Service Management* 16.1 (2019), pp. 27–40. ISSN: 2373-7379. DOI: 10.1109/TNSM.2018.2890273.

[208] S. Ouiazzane, M. Addou, and F. Barramou. “A Multi-Agent Model for Network Intrusion Detection”. In: *2019 1st International Conference on Smart Systems and Data Science (ICSSD)*. Rabat, Morocco, Oct. 2019, pp. 1–5.

[209] P4. *P4 Consortium*. [Online; accessed 31-Dec-2020]. 2020.

[210] C. Papagianni et al. “On the optimal allocation of virtual resources in cloud computing networks”. In: *IEEE Transactions on Computers* 62.6 (2013), pp. 1060–1071. ISSN: 0018-9340. DOI: 10.1109/TC.2013.31.

[211] S. K. Patri et al. “Rational Agent-Based Decision Algorithm for Strategic Converged Network Migration Planning”. In: *IEEE/OSA Journal of Optical Communications and Networking* 11.7 (June 2019), pp. 371–382.

[212] J. Pei et al. “Efficiently Embedding Service Function Chains with Dynamic Virtual Network Function Placement in Geo-Distributed Cloud System”. In: *IEEE Transactions on Parallel and Distributed Systems* 30.10 (2019), pp. 2179–2192. ISSN: 2161-9883. DOI: 10.1109/TPDS.2018.2880992.

[213] V. Petrov et al. “Achieving End-to-End Reliability of Mission-Critical Traffic in Softwarized 5G Networks”. In: *IEEE Journal on Selected Areas in Communications* 36.3 (2018), pp. 485–501. ISSN: 0733-8716. DOI: 10.1109/JSAC.2018.2815419.

[214] Kouvatsos D.D. (eds) Popescu A. Constantinesu D. *On Kleinrock's Independence Assumption In:Network Performance Engineering;Lecture Notes in Computer Science,Berlin, Heidelberg*. Vol. 5233. 2011. DOI: https://doi.org/10.1007/978-3-642-02742-0_1.

[215] *POX Controller*. Accessed: 2020-07-08.

[216] E. S. Pramukantoro et al. “Middleware for Network Interoperability in IoT”. In: *2018 5th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*. 2018, pp. 499–502.

[217] J. Praveen, B. Praveen, and C. S. Ram Murthy. “A First Step Towards Autonomic Optical Burst Switched Networks”. In: *Second International Conference on Autonomic Computing (ICAC'05)*. Seattle, WA, USA, June 2005, pp. 306–307.

[218] Q. Qi et al. “A SDN-based network virtualization architecture with autonomie management”. In: *2014 IEEE Globecom Workshops (GC Wkshps)*. Austin, TX, USA, Dec. 2014, pp. 178–182.

[219] Y. Qiang, Y. Li, and J. Chen. “The Workload Adaptation in Autonomic DBMSs Based on Layered Queuing Network Model”. In: *2009 Second International Workshop on Knowledge Discovery and Data Mining*. Moscow, Russia, Jan. 2009, pp. 781–785.

[220] L. Qu, M. Khabbaz, and C. Assi. “Reliability-Aware Service Chaining In Carrier-Grade Softwarized Networks”. In: *IEEE Journal on Selected Areas in Communications* 36.3 (2018), pp. 558–573. ISSN: 0733-8716. DOI: 10.1109/JSAC.2018.2815338.

[221] L. Qu et al. “Reliability-Aware Service Function Chaining With Function Decomposition and Multipath Routing”. In: *IEEE Transactions on Network and Service Management* (2019), pp. 1–1. ISSN: 2373-7379. DOI: 10.1109/TNSM.2019.2961153.

[222] T. Rahman and S. K. Chakraborty. “Provisioning Technical Interoperability within ZigBee and BLE in IoT Environment”. In: *2018 2nd International Conference on Electronics, Materials Engineering Nano-Technology (IEMENTech)*. 2018, pp. 1–4.

[223] B. Raouf et al. “DeepNFV: A Lightweight Framework for Intelligent Edge Network Functions Virtualization”. In: *Journal of Internet Services and Applications* 9.1 (2018), p. 16. ISSN: 1869-0238. DOI: 10.1186/s13174-018-0087-2.

[224] Jim Reno. *Linux containers vs. VMs: A security comparison*. en. May 2016. (Visited on 08/07/2020).

[225] "Aaron Richard et al. *Autonomous Networks: Empowering Digital Transformation For Telecoms Industry*. Accessed: 08-Jan-2021. May, 2019.

[226] *RISE-6G: Reconfigurable Intelligent Sustainable Environments for 6G Wireless Networks.* Accessed: 19-Oct-2021. 2021.

[227] Y. Rizk, M. Awad, and E. W. Tunstel. “Decision Making in Multiagent Systems: A Survey”. In: *IEEE Transactions on Cognitive and Developmental Systems* 10.3 (May 2018), pp. 514–529.

[228] K. Routh and T. Pal. “A survey on technological, business and societal aspects of Internet of Things by Q3, 2017”. In: *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*. 2018, pp. 1–4. DOI: 10.1109/IoT-SIU.2018.8519898.

[229] M. Ruffini and F. Slyne. “Moving the Network to the Cloud: The Cloud Central Office Revolution and Its Implications for the Optical Layer”. In: *Journal of Lightwave Technology* 37.7 (January 2019), pp. 1706–1716.

[230] *Ryu Controller*. Accessed: 2020-07-08.

[231] C. Sacchi and S. Bonafini. “From LTE-A to LTE-M: a Futuristic Convergence between Terrestrial and Martian Mobile Communications”. In: *2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. 2019, pp. 1–5. DOI: 10.1109/BlackSeaCom.2019.8812825.

[232] G. Sakarkar and V. M. Thakar. “Autonomous software Agent for localization”. In: *2009 International Conference on Intelligent Agent Multi-Agent Systems*. Chennai, India, July 2009, pp. 1–4.

[233] N. Samaan and A. Karmouch. “Towards Autonomic Network Management: an Analysis of Current and Future Research Directions”. In: *IEEE Communications Surveys Tutorials* 11.3 (2009), pp. 22–36. DOI: 10.1109/SURV.2009.090303.

[234] M. G. Samaila et al. “IoT-HarPSecA: A Framework and Roadmap for Secure Design and Development of Devices and Applications in the IoT Space”. In: *IEEE Access* 8 (2020), pp. 16462–16494. DOI: 10.1109/ACCESS.2020.2965925.

[235] S. Sankar and S. Gurumurthi. “Soft Failures in Large Datacenters”. In: *IEEE Computer Architecture Letters* 13.2 (2014), pp. 105–108. ISSN: 1556-6056. DOI: 10.1109/L-CA.2013.25.

[236] S. Schenker. *The future of networking, the past of protocols*. [Online; accessed 01-May-2021]. 2021.

[237] Liron Schiff, Stefan Schmid, and Petr Kuznetsov. “In-Band Synchronization for Distributed SDN Control Planes”. In: *SIGCOMM Comput. Commun.*

[238] S. Schuetz et al. “Autonomic and decentralized management of wireless access networks”. In: *IEEE Transactions on Network and Service Management* 4.2 (November 2007), pp. 96–106. DOI: 10.1109/TNSM.2007.070905.

[239] V. Sciancalepore, F. Z. Yousaf, and X. Costa-Perez. “z-TORCH: An Automated NFV Orchestration and Monitoring Solution”. In: *IEEE Transactions on Network and Service Management* 15.4 (2018), pp. 1292–1306. DOI: 10.1109/TNSM.2018.2867827.

[240] P. Sethi, N. Chauhan, and D. Juneja. “A multi-agent hybrid protocol for data fusion and data aggregation in non-deterministic wireless sensor networks”. In: *2013 International Conference on Information Systems and Computer Networks*. Mathura, India, March 2013, pp. 211–214.

[241] A. H. Shamsan and A. R. Faridi. “Network softwarization for IoT: A Survey”. In: *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*. New Delhi, India, March 2019, pp. 1163–1168.

[242] M. Sharf and D. Zelazo. “Analysis and Synthesis of MIMO Multi-Agent Systems Using Network Optimization”. In: *IEEE Transactions on Automatic Control* 64.11 (March 2019), pp. 4512–4524.

[243] M. Sharf and D. Zelazo. “Network Feedback Passivation of Passivity-Short Multi-Agent Systems”. In: *IEEE Control Systems Letters* 3.3 (May 2019), pp. 607–612.

[244] M. J. Shaw, B. Harrow, and S. Herman. “Distributed artificial intelligence for multi-agent problem solving and group learning”. In: *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*. Vol. iv. Kauai, HI, USA, Jan. 1991, 13–26 vol.4.

[245] Dimitrios Sikeridis et al. “A Comparative taxonomy and survey of public cloud infrastructure vendors”. In: *arXiv preprint arXiv:1710.01476* (2017).

[246] Dimitrios Sikeridis et al. “Context-aware wireless-protocol selection in heterogeneous public safety networks”. In: *IEEE Transactions on Vehicular Technology* 68.2 (2018), pp. 2009–2013.

[247] Dimitrios Sikeridis et al. “Energy-efficient orchestration in wireless powered internet of things infrastructures”. In: *IEEE Transactions on Green Communications and Networking* 3.2 (2018), pp. 317–328.

[248] Dimitrios Sikeridis et al. “Self-adaptive energy efficient operation in UAV-assisted public safety networks”. In: *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE. 2018, pp. 1–5.

[249] Dimitrios Sikeridis et al. "Unsupervised crowd-assisted learning enabling location-aware facilities". In: *IEEE Internet of Things Journal* 5.6 (2018), pp. 4699–4713.

[250] Dimitrios Sikeridis et al. "Wireless powered Public Safety IoT: A UAV-assisted adaptive-learning approach towards energy efficiency". In: *Journal of Network and Computer Applications* 123 (2018), pp. 69–79.

[251] P. Skarin et al. "Towards Mission-Critical Control at the Edge and Over 5G". In: *2018 IEEE International Conference on Edge Computing (EDGE)*. 2018, pp. 50–57. DOI: 10.1109/EDGE.2018.00014.

[252] F. Slim et al. "Towards a dynamic adaptive placement of virtual network functions under ONAP". In: *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. Berlin, Germany, Nov. 2017, pp. 210–215. DOI: 10.1109/NFV-SDN.2017.8169880.

[253] Gan Kim Soon et al. "A Review on Agent Communication Language". In: *Computational Science and Technology*. Ed. by Rayner Alfred et al. Singapore: Springer Singapore, 2019, pp. 481–491. ISBN: 978-981-13-2622-6.

[254] O. Soualah et al. "A Green VNF-FG Embedding Algorithm". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. 2018, pp. 141–149. DOI: 10.1109/NETSOFT.2018.8460013.

[255] O. Soualah et al. "Energy Efficient Algorithm for VNF Placement and Chaining". In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 2017, pp. 579–588. DOI: 10.1109/CCGRID.2017.84.

[256] *Stanford and NASA Launch Tiny IoT Satellites Into Earth's Orbit*. https://theiotmagazine.com/stanford-and-nasa-launch-tiny-iot-satellites-into-earths-orbit-9e5f92487500.

[257] R. Sterritt. "Autonomous and Autonomic Systems: Paradigm for Engineering Effective Software-Based Systems?" In: *2009 33rd Annual IEEE Software Engineering Workshop*. Skovde, Sweden, Oct. 2009, pp. 57–57.

[258] B. L. R. Stojkoska and K. V. Trivodaliev. "A review of Internet of Things for smart home: challenges and solutions." In: *Journal of Cleaner Production* 140.4 (2017), 1454–1464. DOI: 10.1016/j.jclepro.2016.10.006.

[259] J. Strassner. "Autonomic Networking". In: *Fifth IEEE Workshop on Engineering of Autonomic and Autonomous Systems (ease 2008)*. 2008, pp. 3–3.

[260] J. Strassner. "Autonomic networking - theory and practice". In: *2004 IEEE/IFIP Network Operations and Management Symposium (IEEE Cat. No.04CH37507)*. Vol. 1. 2004, 927 Vol.1–. DOI: 10.1109/NOMS.2004.1317811.

[261] J. Strassner and J. O. Kephart. "Autonomic Systems and Networks: Theory and Practice". In: *2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006*. 2006, pp. 588–588.

[262] N. A. M. Subha, G. Liu, and N. H. M. Yusof. "External Consensus in Networked Multi-Agent Systems with Random Network Delay". In: *2018 57th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*. Nara, Japan, Sept. 2018, pp. 427–432.

[263] T. Subramanya and R. Riggio. "Machine Learning-Driven Scaling and Placement of Virtual Network Functions at the Network Edges". In: *2019 IEEE Conference on Network Softwarization (NetSoft)*. Paris, France, June 2019, pp. 414–422. DOI: 10.1109/NETSOFT.2019.8806631.

[264] *Support Vector Machine - Mitosis Technologies*. Accessed on January 5, 2021.

[265] M. Surligas, A. Makrogiannakis, and S. Papadakis. "Empowering the IoT Heterogeneous Wireless Networking with Software Defined Radio". In: *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*. 2015, pp. 1–5.

[266] *System Architecture for the 5G System*. https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/15.03.00_60/ts_123501v150300p.pdf. [Online; accessed 01-May-2021].

[267] M. M. Tajiki et al. "Joint Energy Efficient and QoS-Aware Path Allocation and VNF Placement for Service Function Chaining". In: *IEEE Transactions on Network and Service Management* 16.1 (2019), pp. 374–388. ISSN: 1932-4537. DOI: 10.1109/TNSM.2018.2873225.

[268] M. M. Tajiki et al. "Joint Energy Efficient and QoS-Aware Path Allocation and VNF Placement for Service Function Chaining". In: *IEEE Transactions on Network and Service Management* 16.1 (2019), pp. 374–388. ISSN: 2373-7379. DOI: 10.1109/TNSM.2018.2873225.

[269] H. Tang, D. Zhou, and D. Chen. "Dynamic Network Function Instance Scaling Based on Traffic Forecasting and VNF Placement in Operator Data Centers". In: *IEEE Transactions on Parallel and Distributed Systems* 30.3 (2019), pp. 530–543. ISSN: 2161-9883. DOI: 10.1109/TPDS.2018.2867587.

[270] L. Tang et al. "Queue-Aware Dynamic Placement of Virtual Network Functions in 5G Access Network". In: *IEEE Access* 6 (2018), pp. 44291–44305. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2862632.

[271] Z. Taoqing and H. Xiaoying. "Self-adaptive Network Service Research Based on Multi-Agents in Hierarchy". In: *2010 First International Conference on Networking and Distributed Computing*. Hangzhou, China, Oct. 2010, pp. 314–316.

[272] N. Tcholtchev and R. Chaparadza. "Autonomic Fault-Management and resilience from the perspective of the network operation personnel". In: *2010 IEEE Globecom Workshops*. 2010, pp. 469–474.

[273] *TechEdSat-5 (Technical Education Satellite-5)*. Accessed: 25-Feb-2021.

[274] H. Tianfield. "Multi-agent based autonomic architecture for network management". In: *IEEE International Conference on Industrial Informatics, 2003. INDIN 2003. Proceedings.* Banff, Alberta, Canada, 2003, pp. 462–469.

[275] H. Tianfield. "Towards Edge-Cloud Computing". In: *2018 IEEE International Conference on Big Data (Big Data)*. Seattle, WA, USA, Dec. 2018, pp. 4883–4885.

[276] A. Tizghadam and A. Leon-Garcia. "AORTA: Autonomic network control and management system". In: *IEEE INFOCOM Workshops 2008*. Phoenix, AZ, USA, April 2008, pp. 1–4.

[277] Amin Tootoonchian and Yashar Ganjali. "HyperFlow: A Distributed Control Plane for OpenFlow". In: *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*. USA, 2010, p. 3.

[278] K. Tsagkaris et al. "Customizable Autonomic Network Management: Integrating Autonomic Network Management and Software-Defined Networking". In: *IEEE Vehicular Technology Magazine* 10.1 (February 2015), pp. 61–68.

[279] Weinan William Tseng and et al. "Distributed service subsystem architecture for distributed network management". In: *U.S. Patent* 6,308,207 (2001).

[280] G. Uma, B.E. Prasad, and O.Nalini Kumari. "Distributed intelligent systems: issues, perspectives and approaches". In: *Knowledge-Based Systems* 6.2 (1993), pp. 77 –86. ISSN: 0950-7051. DOI: https://doi.org/10.1016/0950-7051(93)90022-L. URL: http://www.sciencedirect.com/science/article/pii/095070519390022L.

[281] S. van der Meer, J. Keeney, and L. Fallon. "5G networks must be autonomic!" In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. Taipei, Taiwan, April 2018, pp. 1–5.

[282] J. L. Vieira and D. Passos. "An SDN-based access point virtualization solution for multichannel IEEE 802.11 networks". In: *2019 10th International Conference on Networks of the Future (NoF)*. 2019, pp. 122–125.

[283] M. Villamizar et al. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud". In: *2015 10th Computing Colombian Conference (10CCC)*. Accessed: 2021-05-01. 2015, pp. 583–590.

[284] Vishwanath Chukkala et al. "Modeling the radio frequency environment of Mars for future wireless, networked rovers and sensor Webs". In: *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720)*. Vol. 2. 2004, 1329–1336 Vol.2. DOI: 10.1109/AERO.2004.1367731.

[285] Rem W. Collier et al. "MAMS: Multi-Agent MicroServices∗". In: *Companion Proceedings of The 2019 World Wide Web Conference*. WWW '19. San Francisco, USA: Association for Computing Machinery, 2019, 655–662. ISBN: 9781450366755. DOI: 10.1145/3308560.3316509. URL: https://doi.org/10.1145/3308560.3316509.

[286] *W3C, "W3C Semantic Integration Interoperability Using RDF and OWL."* Accessed: 14-Jan-2021.

[287] L. Wang and E. Gelenbe. "Demonstrating Voice over an Autonomic Network". In: *2015 IEEE International Conference on Autonomic Computing*. 2015, pp. 139–140.

[288] L. Wang et al. "Joint Optimization of Service Function Chaining and Resource Allocation in Network Function Virtualization". In: *IEEE Access* 4 (2016), pp. 8084–8094. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2629278.

[289] S. Wang and C. Chang. "Supporting TCP-Based Remote Managements of LoRa/LoRaWAN Devices". In: *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*. 2019, pp. 1–5.

[290] S. Wang et al. "An autonomic communication based conceptual and architecture model for cognitive radio nodes". In: *IET 3rd International Conference on Wireless, Mobile and Multimedia Networks (ICWMNN 2010)*. 2010, pp. 200–204.

[291] W. Wang et al. "Autonomic QoS management mechanism in Software Defined Network". In: *China Communications* 11.7 (2014), pp. 13–23.

[292] Yuxuan Wang et al. "Satellite Edge Computing for the Internet of Things in Aerospace". In: *Sensors 2019*. 20-4375. 2019, pp. 19–20.

[293] Z. Wang et al. "Optimal Control Design for Connected Cruise Control with Edge Computing, Caching, and Control". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2019, pp. 1–6. DOI: 10.1109/INFOCOMWKSHPS47286.2019.9093766.

[294] Z. Wang et al. "Service Function Chain Composition, Placement, and Assignment in Data Centers". In: *IEEE Transactions on Network and Service Management* 16.4 (2019), pp. 1638–1650. ISSN: 2373-7379. DOI: 10.1109/TNSM.2019.2933872.

[295] C. Wannachakrit and T. Anwar. "Development of network management system for network services provider". In: *2011 Malaysian Conference in Software Engineering*. Johor Bahru, Malaysia, Dec. 2011, pp. 439–444.

[296] J. Watada et al. "Emerging Trends, Techniques and Open Issues of Containerization: A Review". In: *IEEE Access* 7 (October 2019), pp. 152443–152472.

[297] J. Wei and S. Cao. "Application of Edge Intelligent Computing in Satellite Internet of Things". In: *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*. 2019, pp. 85–91. DOI: 10.1109/SmartIoT.2019.00022.

[298] T. Werthmann et al. "Task assignment strategies for pools of baseband computation units in 4G cellular networks". In: *2015 IEEE International Conference on Communication Workshop (ICCW)*. 2015, pp. 2714–2720. DOI: 10.1109/ICCW.2015.7247589.

[299] *What is the Deep Space Network?* Accessed: 26-Feb-2021.

[300] *Wi-Fi enables next generation space exploration*. Accessed: 25-Feb-2021.

[301] D. Wu et al. "Towards Distributed SDN: Mobility Management and Flow Scheduling in Software Defined Urban IoT". In: *IEEE Transactions on Parallel and Distributed Systems* 31.6 (2020), pp. 1400–1418. DOI: 10.1109/TPDS.2018.2883438.

[302] M. Wódczak et al. "Standardizing a reference model and autonomic network architectures for the self-managing future internet". In: *IEEE Network* 25.6 (November 2011), pp. 50–56.

[303] G. Xilouris et al. "Towards Autonomic Policy-based Network Service Deployment with SLA and Monitoring". In: *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. Verona, Italy, Nov. 2018, pp. 1–2. DOI: 10.1109/NFV-SDN.2018.8725695.

[304] P. Yadav and S. Vishwakarma. "Application of Internet of Things and Big Data towards a Smart City". In: *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*. 2018, pp. 1–5. DOI: 10.1109/IoT-SIU.2018.8519920.

[305] L. Yala, P. A. Frangoudis, and A. Ksentini. "Latency and Availability Driven VNF Placement in a MEC-NFV Environment". In: *2018 IEEE Global Communications Conference (GLOBECOM)*. 2018, pp. 1–7. DOI: 10.1109/GLOCOM.2018.8647858.

[306] Yan Liu et al. "A model-based approach to adding autonomic capabilities to network fault management system". In: *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*. 2008, pp. 859–862.

[307] B. Yang et al. "Algorithms for Fault-Tolerant Placement of Stateful Virtualized Network Functions". In: *2018 IEEE International Conference on Communications (ICC)*. 2018, pp. 1–7. DOI: 10.1109/ICC.2018.8422444.

[308] H. Yao et al. "The Space-Terrestrial Integrated Network: An Overview". In: *IEEE Communications Magazine* 56.9 (2018), pp. 178–185. DOI: 10.1109/MCOM.2018.1700038.

[309] A. Y. Yazıcıoğlu, M. Egerstedt, and J. S. Shamma. "Formation of Robust Multi-Agent Networks through Self-Organizing Random Regular Graphs". In: *IEEE Transactions on Network Science and Engineering* 2.4 (2015), pp. 139–151.

[310] Q. Ye et al. "End-to-End Delay Modeling for Embedded VNF Chains in 5G Core Networks". In: *IEEE Internet of Things Journal* 6.1 (2019), pp. 692–704. ISSN: 2327-4662. DOI: 10.1109/JIOT.2018.2853708.

[311] H. Yu et al. "Implementation of C-RAN Architecture with CU-CP and CU-UP Separation Based on SDR/NFV". In: *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*. 2019, pp. 1–5.

[312] W. Yu et al. "A Survey on the Edge Computing for the Internet of Things". In: *IEEE Access* 6 (2018), pp. 6900–6919. DOI: 10.1109/ACCESS.2017.2778504.

[313] M. Zeng, W. Fang, and Z. Zhu. "Orchestrating Tree-Type VNF Forwarding Graphs in Inter-DC Elastic Optical Networks". In: *Journal of Lightwave Technology* 34.14 (2016), pp. 3330–3341. ISSN: 0733-8724. DOI: 10.1109/JLT.2016.2565002.

[314] D. Zhang, C. Lu, and X. Jia. "Leader-following consensus of for multi-agent systems under event-triggering communication and network-induced delays". In: *2016 Chinese Control and Decision Conference (CCDC)*. Yinchuan, China, May 2016, pp. 2344–2349.

[315] H. Zhang et al. "A novel autonomic architecture for QoS management in wired network". In: *2010 IEEE Globecom Workshops*. Miami, FL, USA, Dec. 2010, pp. 529–533.

[316] L. Zhang et al. "Performance Analysis and Optimal Cooperative Cluster Size for Randomly Distributed Small Cells Under Cloud RAN". In: *IEEE Access* 4 (2016), pp. 1925–1939. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2550758.

[317] P. Zhang and J. Wang. "SIP based scheme for collaborative autonomic network management". In: *2013 5th IEEE International Conference on Broadband Network Multimedia Technology*. Guilin, China, Nov. 2013, pp. 323–326.

[318] X. Zhang et al. "Virtual Agent Clustering Based Mobility Management Over the Satellite Networks". In: *IEEE Access* 7 (July 2019), pp. 89544–89555.

[319] R. Zhou. "An Online Placement Scheme for VNF Chains in Geo-Distributed Clouds". In: *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. 2018, pp. 1–2. DOI: 10.1109/IWQoS.2018.8624140.