UNIVERSITAT POLITÈCNICA DE VALÈNCIA

INTELLIGENT MULTIMEDIA FLOW TRANSMISSION THROUGH HETEROGENEOUS NETWORKS USING COGNITIVE SOFTWARE DEFINED NETWORKS

Octubre de 2020

Albert Rego Mañez
Supervised by: Jaime Lloret Mauri

Abstract

This thesis addresses the problem of routing in Software Defined Networks (SDN). Specifically, the problem of designing a routing protocol based on Artificial Intelligence (AI) for ensuring Quality of Service (QoS) in multimedia transmissions. In the first part of the work, SDN is introduced. Its architecture, protocols and advantages are discussed. Then, the state of the art is presented, where several works regarding QoS, routing, SDN and AI are detailed. In the next chapter, the SDN controller, which plays the central role in the proposed architecture, is presented. The design of the controller is detailed and its performance compared to another common controller. Later, the routing proposals are described. First, a modification of a traditional routing protocol is discussed. This modification intends to adapt a traditional routing protocol to SDN, focused on multimedia transmissions. Then, the final proposal is described. Its messages, architecture and algorithms are depicted. As regards AI, chapter 5 details the module of the architecture that implements it, along with all the intelligent methods used in the routing proposal. Furthermore, the intelligent route decision algorithm is described and the final proposal is compared to the traditional routing protocol and its adaptation to SDN, showing an increment of the end quality of the transmission. Finally, some applications based on the routing proposal are described. The applications are presented to demonstrate that the proposed solution can work with heterogeneous networks.

Resum

La present tesi tracta el problema de l'encaminament en les xarxes definides per programari (SDN). Específicament, tracta el problema del disseny d'un protocol d'encaminament basat en intel·ligència artificial (AI) per a garantir la qualitat de servici (QoS) en les transmissions multimèdia. En la primera part del treball, s'introdueix les xarxes SDN. Es comenten la seva arquitectura, els protocols i els avantatges. A continuació, l'estat de l'art és presentat, on es detellen els diversos treballs al voltant de QoS, encaminament, SDN i AI. Al següent capítol, el controlador SDN, el qual juga un paper central a l'arquitectura proposta, és presentat. Es detalla el disseny del controlador i es compara el seu rendiment amb altre controlador utilitzat comunament. Més endavant, es descriuen les propostes d'encaminament. Primer, s'aborda la modificació d'un protocol d'encaminament tradicional. Aquesta modificació té com a objectiu adaptar el protocol d'encaminament tradicional a les xarxes SDN, centrat a les transmissions multimèdia. A continuació, la proposta final és descrita. Els seus missatges, arquitectura i algoritmes són mostrats. Pel que fa a l'AI, el capítol 5 detalla el mòdul de l'arquitectura que la implementa, junt amb els mètodes intel·ligents usats en la proposta d'encaminament. A més a més, l'algoritme intel·ligent de decisió de rutes és descrit i la proposta és comparada amb el protocol d'encaminament tradicional i amb la seva adaptació a les xarxes SDN, mostrant un increment de la qualitat final de la transmissió. Finalment, es mostra i es descriuen algunes aplicacions basades en la proposta. Les aplicacions són presentades per a demostrar que la solució presentada en la tesi és dissenyada per a treballar en xarxes heterogènies.

Resumen

La presente tesis aborda el problema del encaminamiento en las redes definidas por software (SDN). Específicamente, aborda el problema del diseño de un protocolo de encaminamiento basado en inteligencia artificial (AI) para garantizar la calidad de servicio (QoS) en transmisiones multimedia. En la primera parte del trabajo, el concepto de SDN es introducido. Su arquitectura, protocolos y ventajas son comentados. A continuación, el estado del arte es presentado, donde diversos trabajos acerca de QoS, encaminamiento, SDN y AI son detallados. En el siguiente capítulo, el controlador SDN, el cual juega un papel central en la arquitectura propuesta, es presentado. Se detalla el diseño del controlador y se compara su rendimiento con otro controlador comúnmente utilizado. Más tarde, se describe las propuestas de encaminamiento. Primero, se aborda la modificación de un protocolo de encaminamiento tradicional. Esta modificación tiene como objetivo adaptar el protocolo de encaminamiento tradicional a las redes SDN, centrado en las transmisiones multimedia. A continuación, la propuesta final es descrita. Sus mensajes, arquitectura y algoritmos son mostrados. Referente a la AI, el capítulo 5 detalla el módulo de la arquitectura que la implementa, junto con los métodos inteligentes usados en la propuesta de encaminamiento. Además, el algoritmo inteligente de decisión de rutas es descrito y la propuesta es comparada con el protocolo de encaminamiento tradicional y con su adaptación a las redes SDN, mostrando un incremento de la calidad final de la transmisión. Finalmente, se muestra y se describe algunas aplicaciones basadas en la propuesta. Las aplicaciones son presentadas para demostrar que la solución presentada en la tesis está diseñada para trabajar en redes heterogéneas.

Acknowledgements

El camino para realizar esta tesis ha sido un largo viaje, que abarca mucho más que los últimos cuatro años. Es por ello que quiero agradecer a todas las personas que han influido en mi educación y formación como persona, empezando por mi familia y, especialmente por mis padres. También agradecer en esa etapa formativa a mis amigos de la infancia y a los maestros con los que desarrollé mi inquietud por los diversos aspectos de la vida. En cuanto a la formación académica, deseo destacar a mis más cercanos amigos de la escuela de ingeniería, con los que he compartido muchos momentos y los cuales me han ayudado en mi formación y junto a los que empecé mi camino profesional. No puedo destacar suficientemente la ayuda de mi supervisor, Ph. D. Jaime Lloret Mauri, que ha sido mi guía durante estos cuatro años. Con él he aprendido a realizar artículos, a revisarlos y proponer mejoras a los autores, a realizar un trabajo tan laborioso como la presente tesis, a empezar con buen pie mi camino en la docencia o a integrarme en un maravilloso equipo de investigación. Y es en ese equipo donde he encontrado el apoyo necesario de mis compañeros. Empezando por gente más veterana como Lorena Parra, Sandra Sendra o José Miguel, los cuales me han aportado una increíble cantidad de ayuda en cada proceso o cada artículo que tenía que afrontar. También, del mismo modo, agradecer a todos los coautores con los que he tenido el placer de colaborar en la producción de artículos científicos: Laura García, Santiago Egea, Oscar Romero y más. Me gustaría destacar la ayuda que me ha prestado Álex Cánovas en el apartado de inteligencia artificial, apartado donde su ayuda ha sido primordial para alcanzar los objetivos propuestos en la tesis. También la ayuda de Juan Ramón, antiguo miembro del grupo, que me ayudó enormemente al inicio del desarrollo del controlador SDN. También agradecer a todas aquellas personas que no he nombrado y que me han brindado apoyo a lo largo de este camino. También a aquellas personas que han sido un refugio en los malos momentos y que me han ofrecido ese apoyo moral que me ha hecho avanzar en los momentos más duros.

Abbreviations

5G	Fifth Generation of Wireless Systems
AAL	Ambient Assisted Living
AI	Artificial Intelligence
ANSI	Ad-Hoc Networking with Swarm Intelligence
AODV	Ad-Hoc On Demand Distance Vector
API	Application Programming Interface
ARA	Ant-Colony-Based Routing Algorithm
ARP	Address Resolution Protocol
AS	Autonomous System
BGP	Border Gateway Protocol
BNC	Big Network Controller
CH	Cluster Head
CNN	Convolutional Neural Network
CPU	Central Process Unit
DRL	Deep Reinforcement Learning
DL	Deep Learning
EIGRP	Enhanced Interior Gateway Routing Protocol
ETArch	Entity Title Architecture
FVSA	Fused Video Surveillance Architecture
GUI	Graphical User Interface
HAS	HTTP Adaptive Streaming
HTTP	Hypertext Transport Protocol
IGRP	Interior Gateway Routing Protocol
IoT	Internet of Things
IP	Internet Protocol
IS-IS	Intermediate System to Intermediate System
JSON	Javascript Object Notation
KPR/I	Key Performance Requirements/Indicators
LLDP	Link Layer Discovery Protocol
LP	Linear Programing
MANET	Mobile Ad-Hoc Network
MDC	Multiple Description Coded
MD-SAL	Model-Driven Service Abstraction Layer
MINA	Multinetwork Architecture Information Architecture
MIoT	Multimedia IoT
ML	Machine Learning
MOS	Mean Opinion Score
MPLS	Multi-Protocol Label Switching
MPLS-TP	Multiprotocol Label Switching - Transport Profile
NB	Northbound
NFV	Network Function Virtualization
NN	Neural Networks

NV	Network Virtualization
NOS	Network Operating System
NQI	New Quality Index
OSHI	Open Source Hybrid IP/SDN
OSPF	Open Shortest Path First
OTT	Over The Top
PPC	Packet Payload Content
PSNR	Peak Signal-to-Noise Ratio
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
RFCP	Route-Flow Control Platform
RIP	Routing Information Protocol
RL	Reinforcement Learning
RTT	Round-Trip Time
SB	Southbound
SDCN	Software Defined Cognitive Networking
SDN	Software Defined Networks
SDN-VQO	SDN Video Quality Orchestrator
SI	Swarm Intelligence
SIR	Sensor Intelligence Routing
SNR	Signal-to-Noise Ratio
SSIM	Structural Similarity Index
TaSDN	Time-aware Software Defined Networking
TaSS	Time-aware Service Scheduling
TCP	Transfer Control Protocol
TLV	Type-Length-Value
UDP	User Datagram Protocol
UML	Unified Marked Language
VANET	Vehicular Ad-Hoc Network
VLC	VideoLAN Client
VM	Virtual Machine
VoIP	Voice over IP
VQAM	Video Quality Assurance Manager
VQM	Video Quality Monitor
WAN	Wide Area Network
Wi-Fi	Wireless Fidelity
WSN	Wireless Sensor Networks

Contents

Abstract **iii**

Contents **xv**

1 Introduction **1**

1.1 Introduction and Architecture 1

1.2 SDN Controllers 5

1.2.1 Opensource Controllers 6

1.2.2 Propietary Controllers 10

1.2.3 Application-oriented Controllers 11

1.3 Applicability and challenges of SDN 13

1.4 Objectives and motivation 15

1.5 Precedents 16

1.6 Thesis Structure 18

2 State of the Art **19**

2.1 Introduction 19

2.2 SDN Controller Studies 21

2.3 Improving QoS or QoE 22

2.4 SDN Applications 29

2.5 Routing in SDN 31

2.6 AI Routing 32

2.7 Conclusion 35

3 Design and Implementation of a Custom SDN Controller **37**

3.1 SDN Background 37
3.1.1 Routing fundamentals 38
3.1.2 OpenFlow Communication and Messages 39
3.1.3 Virtualization in SDN 54
3.2 Architecture 56
3.3 Use Cases 58
3.4 Component and Class Diagrams 64
3.5 Sequence Diagrams and Implementations Fragments 71
3.6 Topology Discovery Process 85
3.7 User Interfaces 86
3.8 Performance Evaluation 93
3.8.1 Linear Topology 93
3.8.2 Tree Topology 98
3.8.3 Torus Topology 103
3.9 Conclusion 108

4 Routing Proposal 111
4.1 Introduction 111
4.2 Dynamic OSPF Adaptation 112
4.2.1 Routing Proposal 112
4.2.2 Metric Analysis 116
4.2.3 Messages 119
4.2.4 Algorithm and Process 120
4.2.5 Methodology and Results 124
4.2.6 Comparison with other OSPF variations 132
4.2.7 Conclusion 134
4.3 Intelligent SDN Routing Protocol 136
4.3.1 Aim and scope 137
4.3.2 Data Structures 138
4.3.3 States and algorithm 140
4.3.4 Messages 146
4.4 Conclusion 152

5 Artificial Intelligent Module 155
5.1 Introduction and architecture 155
5.2 Video Transmission Study 156
5.2.1 Development and Analysis of QoS Results in Video Transmission . 157
5.2.2 Development and Analysis of QoE Results in Video Transmission . 161
5.2.3 Data Preprocessing 165
5.3 QoS Degradation Estimator 167
5.3.1 Database Pre-processing 168

5.3.2 QoE Estimation Model 169
5.3.3 Classification Model of the Type of Traffic 178
5.3.4 Network Architecture of the Type of Traffic Classification Model . 182
5.3.5 Classification and Modelling of Network and Video for QoE Regulation 189
5.4 Robust Multimedia Traffic Management System 196
5.4.1 Network Architecture 196
5.4.2 Simple Management Algorithm 197
5.4.3 Robust Management System Implementation 200
5.4.4 Test Bench 202
5.4.5 Conclusion 207
5.5 A Proposed Machine Learning System to Solve Multimedia Transmission Problems 208
5.5.1 Architecture 209
5.5.2 Network Considerations 210
5.5.3 Reinforcement Learning Algorithm 211
5.5.4 Results 226
5.5.5 Conclusion and Future Work 238
5.5.6 Reward Update System 240
5.6 Routing Alternative Evaluation 241
5.6.1 Scenario 1 244
5.6.2 Scenario 2 247
5.6.3 QoE Results 250
5.7 Conclusion 252

6 Applications 253

6.1 Introduction 253
6.2 SDN Applied to Smart Irrigation 253
6.2.1 Urban Scenarios 254
6.2.2 Employed Sensors 255
6.2.3 Architecture 256
6.2.4 Messages 260
6.2.5 Algorithm 265
6.2.6 Requirements for the implantation of the proposal 266
6.2.7 Conclusions 268
6.3 SDN for Video Surveillance 269
6.3.1 Architecture 270
6.3.2 Algorithm 272
6.3.3 Messages 273
6.3.4 Methodology and Results 276
6.4 SDN for Emergency Managemenet in Smart Cities 282

6.4.1 Architecture and Algorithm 282
6.4.2 Messages 284
6.4.3 Energy Consumption Model 288
6.4.4 Methodology and Results 292
6.4.5 Conclusion 298
6.5 Conclusion 300

7 Conclusions 305
7.1 Introduction 305
7.2 Conclusion 305
7.3 Problems Faced and Personal Contributions 308
7.4 Future Research 309
7.5 Publications derived from the PhD Thesis 310

References 313

List of Figures

1.1 Internet Users (Millions, data extracted from [1]). 2
1.2 Global subscribers among video services (data extracted from [1]). 2
1.3 Network planes of a) traditional networks and b) SDN. 4
1.4 Interaction between the SDN controller and the OpenFlow-enabled switches. 5

3.1 Handshake communication process. 53
3.2 Network architecture and interfaces of an SDN. 56
3.3 Custom SDN Controller Architecture. 57
3.4 User case diagram 59
3.5 Component Diagram. 69
3.6 Class Diagram. 72
3.7 "Add node" sequence diagram. 73
3.8 "Test connectivity" sequence diagram. 74
3.9 "Ask Action" sequence diagram. 75
3.10 "Load Topology" sequence diagram. 76
3.11 "Load Topology" JSON. 76
3.12 "Load Topology" parsing code fragment. 77
3.13 "Send Flow" sequence diagram. 78
3.14 "Flow Mod" implementation fragment. 79
3.15 "Perform Action" sequence diagram. 80
3.16 "Create Flow Mod" sequence diagram. 81
3.17 "Load Changes" sequence diagram. 82
3.18 "Load Changes" JSON. 83
3.19 Request Stats sequence diagram. 84
3.20 Get Stats sequence diagram. 85

3.21 Main interface. 87
3.22 Main interface with a list of nodes. 88
3.23 Send Flow Mod interface. 89
3.24 Load Changes interface. 90
3.25 Load File window. 91
3.26 Load Topology interface. 92
3.27 Statistics Manager interface. 92
3.28 Linear Topology. 93
3.29 Minimum RAM used in linear topology. 94
3.30 Average RAM used in linear topology. 95
3.31 Maximum RAM used in linear topology. 96
3.32 Minimum CPU used in linear topology. 97
3.33 Average CPU used in linear topology. 97
3.34 Maximum CPU used in linear topology. 98
3.35 Tree Topology. 99
3.36 Minimum RAM used in tree topology. 100
3.37 Average RAM used in tree topology. 100
3.38 Maximum RAM used in tree topology. 101
3.39 Minimum CPU used in tree topology. 102
3.40 Average CPU used in tree topology. 102
3.41 Maximum CPU used in tree topology. 103
3.42 Torus Topology. 104
3.43 Minimum RAM used in torus topology. 105
3.44 Average RAM used in torus topology. 105
3.45 Maximum RAM used in torus topology. 106
3.46 Minimum CPU used in torus topology. 107
3.47 Average CPU used in torus topology. 107
3.48 Maximum CPU used in torus topology. 108

4.1 Elements and network of the proposed architecture. 112
4.2 Routing protocol structure. 113
4.3 New routing protocol structure. 114
4.4 Cost per delay and loss rate with a bandwidth of 10Mbps. 117
4.5 Cost per delay and loss rate with a bandwidth of 100Mbps. 118
4.6 Cost per delay and loss rate with a bandwidth of 1000Mbps. 119
4.7 Messages structure. 120
4.8 Initial message exchange. 122
4.9 Message exchange when a new packet arrives 123
4.10 Message exchange when the AI module updates the different factors used in the metric. 124
4.11 Algorithm diagram message exchange. 125

4.12 Topology used in the experiments. 126
4.13 Bandwidth in bits per second registered in scenario 1. 128
4.14 Delay in ms registered in scenario 1. 129
4.15 Jitter in ms for each packet in scenario 1. 130
4.16 Delay in ms obtained in scenario 2 for each packet 132
4.17 Jitter in ms for each packet in scenario 2. 134
4.18 Bandwidth in bits per second in scenario 2. 135
4.19 Delay in ms obtained in scenario 3 for each packet. 136
4.20 Jitter in ms for each packet in scenario 3. 137
4.21 Delay in ms obtained in scenario 4 for each packet. 138
4.22 Jitter in ms for each packet in scenario 4. 139
4.23 Loss rate presented with OSPF and the proposal in scenario 4. 140
4.24 Bandwidth consumed for each transmission, by using the same metrics for each flow and by using different metric for the different flows. 141
4.25 Jitter obtained in scenario 5. 142
4.26 Module tasks. 144
4.27 Routing Module submodules. 145
4.28 Data exchange between modules. 146
4.29 States Diagram. 148
4.30 RTG_SET_ACTIONS message. 150
4.31 RTG_NEW_ACTION message. 151
4.32 RTG_SET_ACTIONS message. 152
4.33 RTG_FLOW message. 153
4.34 RTG_MULTIMEDIA message. 154
4.35 RTG_PERFORM_ACTION” message. 154

5.1 Elements and network of the proposed architecture. 157
5.2 Delay with loss rate from 0% to 1% and different resolutions with 30fps. . . 159
5.3 Number of lost packets with different resolutions with 30fps. 160
5.4 Average of lost packets with different resolutions with 30fps. 161
5.5 Jitter in ms with different resolutions with 30fps. 162
5.6 Delay with loss rate from 0% to 1% and a 1920x1080 30fps video streaming. 163
5.7 Delay in the same conditions as the previous case but also with 60fps videos. 164
5.8 Delay depending on jitter, from 10ms to 60ms, and video resolution with 30fps streaming. 165
5.9 Number of lost packets depending on jitter, from 10ms to 60ms, and video resolution with 30fps streaming. 166
5.10 Average of lost packets depending on jitter, from 10ms to 60ms, and video resolution with 30fps streaming. 167
5.11 Jitter depending on jitter, from 10ms to 60ms, and video resolution with 30fps streaming. 168

5.12 Delay obtained in a 10Mb/s network with 800x600 30fps disturbing traffic. 169
5.13 Delay obtained in a 100Mb/s network with 1920x1080 30fps disturbing traffic. 170
5.14 Delay obtained in a 100Mb/s network with 1920x1080 30fps disturbing. . . 171
5.15 Number of packets lost in a 10Mb/s network with 800x600 30fps disturbing traffic. 172
5.16 Average of lost packets obtained in a 10Mb/s network with 800x600 30fps disturbing traffic. 173
5.17 Average of lost packets obtained in a 10Mb/s network with 1920x1080 30fps disturbing traffic. 174
5.18 Dataset related to the subjective measurements after analyzing both, a 30fps and a 60fps video streaming. 175
5.19 Preprocessing based on sampling for each frame. 176
5.20 Preprocessing based on sampling the lost packets each time instant. 177
5.21 Preprocessing based on sampling jitter and delay each time instant. 178
5.22 Learning results for QoE estimation in the different classification models according to the confusion matrix. 179
5.23 Learning results for QoE estimation in the different regression models according to the confusion matrix. 180
5.24 Test results of the QoE estimation for CSVM and RNN models. 181
5.25 Training results for multimedia traffic classification obtained from the different classification models according to the confusion matrix. 182
5.26 Training results for multimedia traffic classification obtained from the different regression models according to the confusion matrix. 183
5.27 Test results of the multimedia traffic classification of CSVM and RNN models. 184
5.28 General architecture of the neuronal network used in this work. 185
5.29 Mean and standard deviation of frame rate and minimum frame rate with regard to MOS. 191
5.30 Mean and standard deviation of width and height with regard to MOS. . . 191
5.31 Mean and standard deviation of bitrate and stream size with regard to MOS. 192
5.32 Mean and standard deviation of bandwidth with regard to MOS. 192
5.33 Subpatterrns for MOS = 4. 194
5.34 Subpatterrns for MOS = 5. 194
5.35 Network architecture example. 198
5.36 SDN Architecture used in this system explanation. 198
5.37 Management system algorithm. 199
5.38 Hardware used in the test bench. 203
5.39 Video characteristic selection performed by the system according to the NN model (based on QoE estimation) to solve the critical situation with all the subpatterns. 205
5.40 Results of the statistical study relationship jitter and delay with every video characteristic selection. 205

5.41 Video characteristic selection performed by the system according to the original model (subjective QoE) to solve the critical situation without all the subpatterns. 206
5.42 Results of the statistical study relationship jitter and delay on the second test with every video characteristic selection. 207
5.43 SDN Controller Modules. 210
5.44 Network Example. 211
5.45 Diagram of the states of a multimedia flow in the system. 214
5.46 Example of Array H. 215
5.47 Example of Array M. 216
5.48 γ factor value on each timestamp t. 218
5.49 OFP_FLOW_MOD message structure. 220
5.50 General AI algorithm. 223
5.51 Reward calculation algorithm. 224
5.52 Network tested. 226
5.53 Equipment characteristics. 227
5.54 H array used in the test. 228
5.55 Bandwidth evolution in the first test (τ_{rew} from 10 to 40). 229
5.56 Bandwidth evolution in the first test (τ_{rew} from 50 to 80). 230
5.57 Delay evolution in the first test (τ_{rew} from 10 to 40). 230
5.58 Delay evolution in the first test (τ_{rew} from 50 to 80). 231
5.59 Jitter evolution in the first test (τ_{rew} from 10 to 40). 232
5.60 Jitter evolution in the first test (τ_{rew} from 50 to 80). 233
5.61 Loss rate for each scenario. 234
5.62 MOS obtained in each scenario. 235
5.63 Bandwidth evolution in the second test. 236
5.64 Delay evolution in the second test. 237
5.65 Jitter evolution in the second test. 237
5.66 Loss rate obtained in the second test. 238
5.67 MOS gathered for the second test. 239
5.68 Proposed topology. 242
5.69 Loss rate obtained in the second test. 242
5.70 Actions selected by the RL algorithm in scenario 1. 244
5.71 Bandwidth consumed in scenario 1. 245
5.72 Delay presented in scenario 1. 246
5.73 Jitter presented in scenario 1. 246
5.74 Actions selected by the RL algorithm in scenario 2. 247
5.75 Bandwidth consumed in scenario 2. 248
5.76 Delay presented in scenario 2. 249
5.77 Jitter presented in scenario 2. 249
5.78 Packet losses. 250

5.79 QoE results. 251

6.1 Rain sensor. 256
6.2 Soil moisture sensor. 257
6.3 Temperature sensor. 258
6.4 Solar radiation sensor. 259
6.5 Presence sensor. 259
6.6 Architecture of the proposed system. 261
6.7 Example of multiple networks connected in the Smart City. 262
6.8 Layer Architecture. 263
6.9 Messages with non-empty value field and their structure and length. . . . 264
6.10 Algorithm of the irrigation process. 267
6.11 Data sent in each scenario to the SDN. 269
6.12 Scheme of the architecture proposed. 270
6.13 Scheme of each actor in the system and their interaction. 271
6.14 SDN controller operation diagram. 276
6.15 Message communication process when a new packet arrives. 277
6.16 Message communication process when statistics are demanded. 278
6.17 Custom messages used when actions in the IoT networks are required. . . . 279
6.18 Message communication process when there is an SDN-node-based action. . 280
6.19 Message communication process when the IoT network is affected. 281
6.20 Topology used in the experiments. 282
6.21 Bandwidth obtained in scenario 1. 283
6.22 Jitter obtained in scenario 1. 284
6.23 Bandwidth obtained in scenario 2. 285
6.24 Jitter obtained in scenario 2. 286
6.25 Bandwidth obtained in scenario 3. 287
6.26 Jitter obtained in scenario 3. 288
6.27 Loss rate obtained in scenario 3. 289
6.28 Bandwidth obtained in scenario 4. 290
6.29 Jitter obtained in scenario 4. 291
6.30 QoE obtained in each scenario. 292
6.31 Scenario and proposed architecture. 293
6.32 Message exchange when alarm is detected. 295
6.33 Message exchange when alarm is finished. 296
6.34 Message Structure. 297
6.35 Scheme of the city used for testing. 298
6.36 SDN corresponding to the Smart City. 299
6.37 Delay in normal traffic sending without system application. 300
6.38 Delay in normal traffic sending with system application. 301
6.39 Delay in emergency traffic without system application. 301

6.40 Delay in emergency traffic with system application. 302
6.41 Energy consumed per node. 302
6.42 Total energy consumed per scenario with 2 neighbors. 303
6.43 Total energy consumed per scenario with 4 neighbors. 303
6.44 Total energy consumed per scenario with 6 neighbors. 304

List of Tables

3.1 Fields used to match flow entries. 39
3.2 Structure of the OpenFlow header. 40
3.3 Structure of the OpenFlow features reply message. 42
3.4 Description of the capabilities bitmap. 43
3.5 Physical port structure. 43
3.6 Structure of configuration message. 44
3.7 Possible values of the flag field. 44
3.8 Structure of the flow modification message. 45
3.9 Possible actions to be performed by a datapath after a match. 46
3.10 Port modification message. 46
3.11 Possible values of the type field on a statistic message. 47
3.12 Structure of the body of the description statistics message reply. 47
3.13 Structure of the body of the flow statistics message reply. 48
3.14 Structure of the body of the flow statistics message reply. 49
3.15 Structure of the body of the table statistics message reply. 49
3.16 Structure of the body of the port statistics message reply. 50
3.17 Structure of the body of the queue statistics message reply. 51
3.18 Structure of the body of the packet-in message. 51
3.19 Structure of the body of the packet-out message. 52
3.20 Structure of the error message. 52
3.21 Details of Add node user case. 60
3.22 Details of Test Connectivity node user case. 60
3.23 Details of Ask Action user case. 61
3.24 Details of Stats Request user case. 62
3.25 Details of Load Topology user case. 63

3.26 Details of Get Stats user case. 64
3.27 Details of Load Changes user case. 65
3.28 Details of Create Flow Mod user case. 66
3.29 Details of Send Flow Mod user case. 67
3.30 Details of Perform Action user case. 68

4.1 Scenarios used for testing. 127
4.2 Results of each scenario using OSPF and our proposal. 131
4.3 Comparison of OSPF modifications. 133
4.4 Data structures used by the routing module. 143
4.5 Values for type and len of each message. 149

5.1 Optimization of the RNN classification model. 189
5.2 Comparison of the proposals. 225
5.3 Bandwidth statistics for each scenario. 229
5.4 Delay statistics for each scenario. 231
5.5 Jitter statistics for each scenario. 233
5.6 Relevant metrics for action evaluation. 241
5.7 Networks between switches. 243
5.8 Equipment specifications. 243
5.9 Software and video used. 244
5.10 Users data. 251

6.1 Possible values gathered by each sensor. 260
6.2 Data quantity and type sent by each kind of node. 264
6.3 Data length of each data type. 265
6.4 Number of nodes for different purposes in each scenario. 268
6.5 Possible actions and their category. 273
6.6 Actions taken when each message is sent. 275

Chapter 1

Introduction

1.1 Introduction and Architecture

Nowadays, networks are used for different applications. These applications require network resources of different nature. For example, multimedia applications do not require the same resources as massive downloads or applications in Internet of Things (IoT) environments.

In the last years, multimedia traffic has growth significantly. In fact, according to Cisco's estimations, the Internet video streaming and downloads present a positive growth and will grow to more than 80% of all consumer Internet traffic by 2022 [1]. With the development of devices such as smartphones, tablets, smart TVs and technologies like IoT or wireless sensor networks (WSNs) the number of devices in Internet has been increased [1]. Services like YouTube o Netflix are widely used around the world [2]. This is due to the rapid growth of internet users [3]. As Fig. 1.1 shows, the continent with the biggest number of internet users is Asia (2,062 million of users) followed by Europe with 705 million of internet users. On the other hand, platforms as Netflix, HBO and Amazon Prime video services are becoming as the main medium to consume multimedia content (see Fig. 1.2).

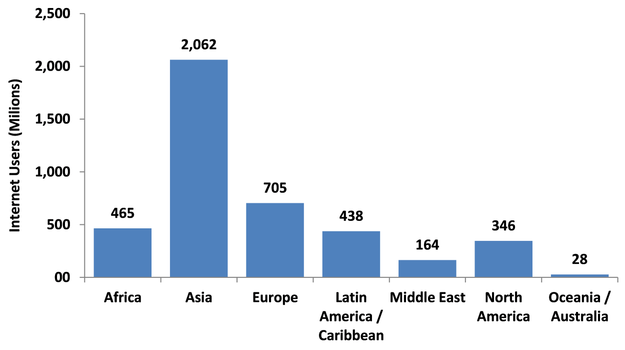


Figure 1.1: Internet Users (Millions, data extracted from [1]).

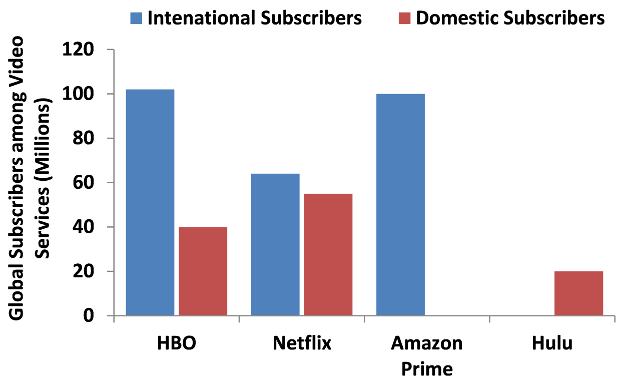


Figure 1.2: Global subscribers among video services (data extracted from [1]).

Moreover, not only video on demand, but also other numerous applications like video conference, Voice over IP (VoIP) or gaming require the transmission of multimedia content through the network. Multimedia content has different network resource requirements than other types of content. Ensuring QoS [4] and Quality of Experience (QoE) [5] is one of the main concerns of researchers that look for novel ways to guarantee good quality video transmission. The quantity of bandwidth used by this kind of traffic, and the necessity of ensuring the QoS, has caused researchers to propose new types of network technologies. The delay in the communication and the packet loss must be minimized in order to obtain the greatest QoE for the end user.

In a network composed by several computers, we can classify the different elements attending to the role these devices have in the transmission process. Hence, the network can be divided into three different layers:

- Application layer: it provides the different services running in the network.
- Control layer: it is composed by the logical processes. It determines which management actions should be performed with the data. The routing and switching processes are located in this layer.
- Data layer: it forwards the traffic through the different ports of a switch or a router.

In the traditional networks, the network architecture follows a plane distribution like the one in The network nodes redirect the data through the different ports and routes. This is called the data plane. However, every node in the network takes the decision of which action perform when a packet arrives. This logical process, composed by a set of rules, which usually follows the policies of a set of network protocols, is called control plane. The network nodes, routers and switches for instance, in a traditional network architecture must decide the best route for the traffic, follow security policies and discard some packets. Therefore, the nature of the network is fully distributed, because all the logic of the network is located in each node.

SDN is a new paradigm that changes the architecture of the network, separating the control plane, which is located now in a central entity, from the data plane [6, 7].

SDNs introduce this central entity called SDN controller [8]. The controller is the central element in a virtualized network that manages and controls the network devices, having a centralized view and communicating with the different layers in order to operate in a more efficient way. The SDN controller manages all the control procedures of the network. These logical procedures that control the work of the network are now located in the SDN controller, such as routing and switching protocols, security policies, network address translation and so on. Therefore, the entire control logic of the network is placed into this central entity. This logic is known as control plane. The network nodes become mere points of the network where the data is forwarded or discarded. In an SDN, they only perform the data plane function. This new architecture is depicted in Fig. 1.3b.

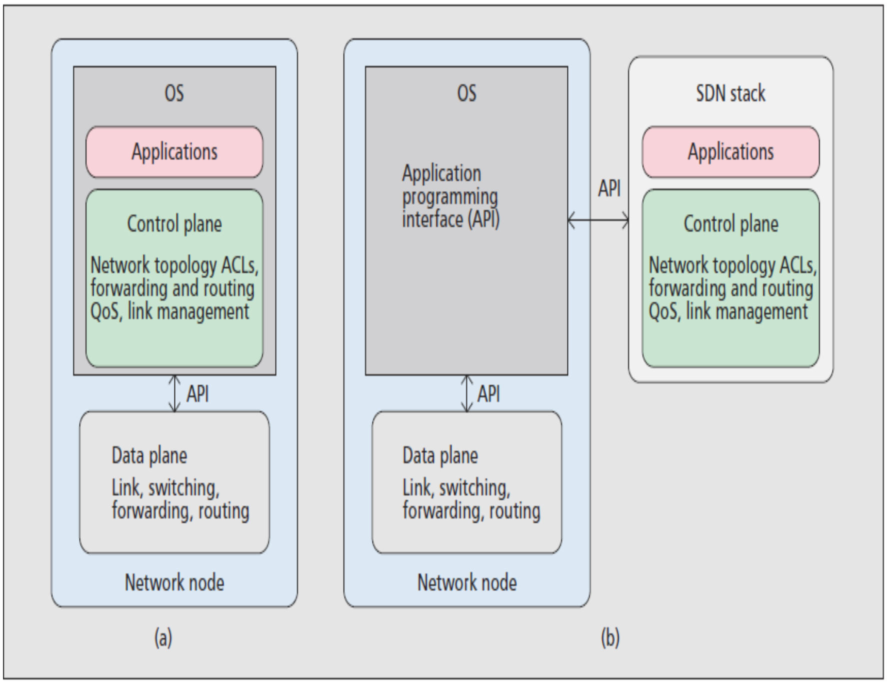

OS
Applications
Control plane
Network topology ACLs, forwarding and routing QoS, link management
API
Data plane
Link, switching, forwarding, routing
Network node
(a)
OS
Application programming interface (API)
API
API
Data plane
Link, switching, forwarding, routing
Network node
SDN stack
Applications
Control plane
Network topology ACLs, forwarding and routing QoS, link management
(b)

Figure 1.3: Network planes of a) traditional networks and b) SDN.

Introducing this entity and moving forward a centralized network has some consequences. Firstly, this controller has to establish a continuous communication with the network nodes. This communication is needed to inform about the rules and policies the controller has established. For instance, as regards routing, the changes in the path that the SDN controller decides must be implemented in the network nodes. Since the network nodes of an SDN merely perform operations with the data (like forwarding, performing modifications in the packet and so on), the rules must be communicated from the SDN controlled. This interaction is performed through the southbound interface or SB API. This interface, which is displayed in Fig. 1.4 , connects the network nodes with the SDN controller. The communication is based on the OpenFlow protocol [9]. OpenFlow is a standard communication protocol that allows the communication between the SDN controller and the network nodes. In this work, the OpenFlow version used is 1.0. Consequently, when it is not commented, the version of the standard discussed will be 1.0.

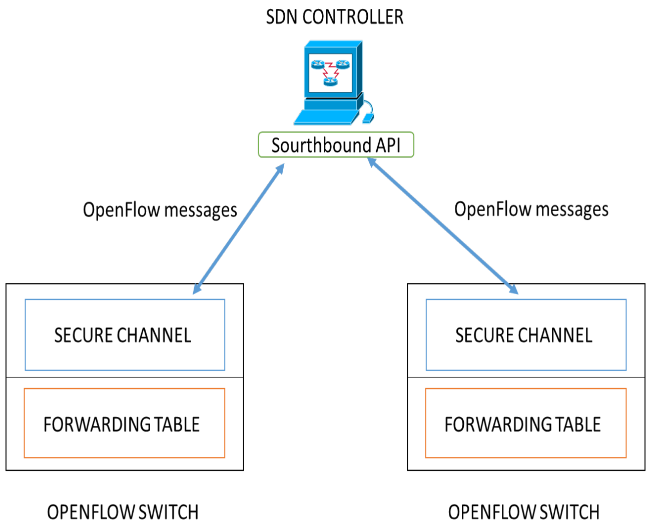


Figure 1.4: Interaction between the SDN controller and the OpenFlow-enabled switches.

Secondly, moving out the control plane from the network nodes allow defining this entity as it is needed. The SDN controller is a software entity. This software entity can be run in a PC. This allows bringing in new applications into the network. Furthermore, the flexibility the SDN controller has to manage the network and adapt to the network problems is much deeply that the traditional control plane. This characteristic provides the SDN controller interact with software applications to implement functionalities that have never been seen in the traditional networks. This interaction is performed through the northbound API or NB API.

1.2 SDN Controllers

Many SDN Controllers with different characteristics have been developed in the last years.

A brief description and comparison of the most often used is presented next. Firstly, the commonly used open-source controllers are described. After that, the proprietary controllers, the ones made and sell by the manufacturers, are commented. Finally, the application-oriented controllers are briefly discussed.

1.2.1 Opensource Controllers

NOX

NOX [10] is the first controller that appeared in SDN. Developed by Niciria Networks, now owned by VMware [11]. Introduced in 2009, the project was forked and presented different lines of development. The original version, known as NOX classic [12], the new NOX [13], and POX. POX is described in the next subsection. As regards the second version of NOX, it was completely built in C++, designed and implemented in a more solid base that its predecessor. However, the number of applications is lower than the available for the first version.

The aim of NOX is to provide a framework to develop new network applications. Therefore, the developers and researchers can innovate and create novel software-based solutions for enterprise networks. The core of NOX provides methods to help developers, such as event handler, threading and network packet processing. Moreover, it also has OpenFlow APIs to make easier the interaction between the SDN controller and the switches.

In NOX, all the applications are programmed as components. They inherit from the component class. Consequently, the applications are built from a set of components that cooperate to implement a desired functionality. Each component encapsulates a part of the functionality. NOX is compatible with some Graphical User Interface (GUI). NOX was the base to show the potential of SDN. It was the original SDN controller. However, it has been replaced for more modern SDN controllers, starting from its successor, POX.

POX

POX [14] was born from NOX. It is an open-source development platform oriented to SDN control applications. It is Python-based, unlike NOX, which allowed C++ applications. The POX controllers present an OpenFlow interface called Pythonic, to communicate with the switches. Like NOX, it is based on modules, which are called components and can be reused. For instance, POX present some sample components for topology discovery, path selection or layer-two switches, among others. It is capable to run in any operating system. POX can support the same GUI as NOX and it can provide better performance as compared to NOX written in Phyton. For these reasons, POX replaced NOX as the base of developing SDN applications.

Beacon

POX and NOX are not the only projects to develop an open-source SDN controller. Beacon [15], currently deprecated, is a Java-based Openflow controller presented in Stanford [16]. Their aim was to develop an SDN controller that was able to run and stop applications at runtime and that had high performance.

Beacon presents a multithread SDN controller designed from a shared queue. Each switch is assigned to a single I/O thread. These I/O threads are the ones that read and the OpenFlow messages from switches. Moreover, they also write the outgoing OpenFlow messages that the SDN controller sends to the switches. With this design, there must be another set of threads, the consumers, which take OpenFlow messages out from the shared queue.

As regards the northbound API, Beacon uses the Spring library to implement REST API. Any application that is written for the beacon controller must use this REST API to implement its functionality in the network. The modular approach of Beacon controller allows the users and developers to disable the modules they do not use.

Maestro

It was presented by a research group of the department of computer science at Rice University [17]. Maestro is a multi-threaded Java-based SDN controller. It focuses on the OpenFlow specific characteristics to achieve the best impact in deployed OpenFlow networks.

Maestro manages the OpenFlow messages writing and reading at low level. It manages the TCP connections and translate the raw OpenFlow messages into high level data structures.

The applications in Maestro are high level functionalities that are implemented in modules. Programmers can flexibly modify the behavior of the available applications. However, they can also add new applications to provide novel functionality to the network.

Floodlight

Floodlight [18] uses Beacon as its base. While the use of Beacon has been reduced to the minimum, Floodlight is one of the most popular open-source SDN controllers and the most popular Java-based, at least until the irruption of OpenDayLight.

Floodlight was a contribution of Big Switch Networks [19]. This company had a commercial SDN controller called Big Network Controller (BNC). The architecture of Floodlight is based on BNC. It is a modular architecture where each module implements a specific functionality. The different functionalities can be classified into four different groups, as follows:

- Controller Modules: The services the northbound API use and the ones that define the possibilities of the controller. Some of these services are: the link discovery manager, the provider (which translate OpenFlow messages into events), the topology service (based on the result of the link discovery manager) or the packet forwarder.
- Floodlight Services: The services that provides functionality to other modules or to the REST API. They must implement an IFloodlightService. Some of these services are the firewall, the debug event or the access control list.
- Built-in Module Applications: Some applications are included in the distribution of Floodlight. Not only Floodlight, but most SDN controllers include this kind of applications. They are a good starting point for developers to understand how to create new modules.
- REST-API based applications: The applications developed for the network interact with the controller through a REST-API.

Write a module, however, it is not a simple task. Fortunately, Floodlight is well documented [20]. The module must define the events, the messages and the utilities that compose its functionality. The developer must create a listener. After implementing the module, and setting up the dependencies, it must be registered.

OpenDayLight

Released in 2013 by The Linux Foundation [21], OpenDayLight is an open-sourced SDN platform for controlling and customizing SDN [22]. Based on a modular design, the network administrators or developers can build the controller adapted to their needs.

The architecture of OpenDayLight is Model-Driven [23]. All the network devices or applications are represented as models, or objects. The Model-Driven Service Abstraction Layer (MD-SAL) processes the interactions between the models.

The models are represented by the YANG data model language [24]. This allows the representation of the capabilities of the network applications, or the description of the network devices, without the requirement of knowing the specific implementation.

OpenDayLight works with the consumer-producer architecture. A producer model implements an API and a consumer uses this API and consumes its data. The MD-SAL is the framework that allows this architecture and this interaction between the models. Therefore, a consumer can find a provider that it is interested in. While the producer creates notifications, the consumer receives them. The consumers read the data from the storage, which have been written by a producer.

OpenDayLight became the most widely deployed open-source SDN controller. It is one of the most widely used in researches and it is the base of other frameworks such as ONAP [25], OpenStack [26] and OPNFV [27].

Ryu

Ryu [28] is a component-based open-source software defined networking framework. It provides software modules, called components, so that developers can easily create network control and management applications.

Ryu is based and fully written in Python. The components provide a defined API exposed to the developers to use it in their own applications [29]. One of the characteristics of Ryu is that allows using several protocols for its southbound API.

The architecture of Ryu is based on a set of libraries for the southbound API, which allows that compatibility with the several protocol. Ryu has a controller module that interacts with the southbound API to manage the network devices. The Ryu manager is the main executable. It listens to the connections from the switches. In addition, the core-processes provide event management, state reading and management, messaging, memory management, among other.

Ryu has an app manager module. This is the module which allows the interaction between the core of Ryu and the applications. All applications inherit from the app manager. The Ryu northbound API it uses REST so that the applications can access to the API and use the core services.

Trema

Trema [30] is a full-stack, easy-to-use framework for developing OpenFlow controllers in Ruby and C. It allows the users to build and configure freely an SDN controller. The support of both programming languages, with such a huge difference between them, gives the user the freedom to choose depending on their knowledge and the performance requirements.

Trema architecture includes several core modules for packet filtering or switch management. Furthermore, it has an OpenFlow protocol interface, a set of libraries to perform message, events, data and core operations. Finally, Trema has a monitor module and an emulator for internal development.

1.2.2 Propietary Controllers

Some companies have developed their own SDN controllers. Some of them are based on open-source controllers. In this subsection, a brief description of two different proprietary controllers is presented.

CISCO Open SDN Controller

The SDN controller of Cisco is a commercial distribution of OpenDayLight. Currently, the controller is no longer being sold.

The Cisco Open SDN Controller [31] provided the Cisco clients with a ready-to-use control software to manage and monitor the SDN. Due to the fact it was an OpenDayLight derivate product, the controller was updated in base of the OpenDayLight software updates of the community.

Some main features and capabilities of the controller were its scalability, the monitoring, metrics collection and log management. Furthermore, it allows

REST API in the northbound API and a Java API to create functions to deliver custom capabilities [32]. However, the greatest feature of the CISCO OPEN SDN controller was the Open Virtual Appliance packaging. It provided a simplified installation and deployment flexibility.

ARUBA VAN SDN Controller

Aruba VAN SDN controller [33] is the proprietary SDN controller of HP. The VAN SDN controller provides Aruba clients with a software solution for network management, provisioning and orchestration.

The controller provides open APIs so that developers can create novel network applications. For that purpose, developers can use either RESTful API or Java programs. As regards southbound API, the controller allows up to 50 OpenFlow-enabled HP switch models. The OpenFlow protocol versions supported by the VAN SDN Controller are 1.0 and 1.3.

Some features of the controller are: high availability and scalability, security with HTTPS and TLS, a GUI and the use of REST API or native Java OSGI bundles to develop applications. Nonetheless, like CISCO OPEN SDN controller, the main advantage is that the controller is ready to use or to install without great effort. Aruba VAN SDN controller even provides a virtual machine deployment option to deploy the controller with minimal effort.

1.2.3 Application-oriented Controllers

In this subsection, a brief description of some application-oriented controllers is provided. These controllers are specifically control software focused on provide a specific network functionality.

RouteFlow

RouteFlow [34] is a source project whose aim is to provide virtualized IP routing services in an SDN. It is based on the OpenFlow protocol.

The platform uses an OpenFlow controller application, an independent RouteFlow server and a virtualized network [35]. In these virtualized networks, RouteFlow reproduces the connectivity of a traditional physical infrastructure. RouteFlow uses routing engines to generate the forwarding information. The forwarding information base is generated according to the selected routing protocols and collected by the RouteFlow client. Then, this info is translated to the OpenFlow-enabled devices in the SDN.

RouteFlow is then a software control framework to implement traditional routing algorithms into SDN. However, it is not the only application-oriented control software in SDN.

FlowVisor

FlowVisor [36] is an OpenFlow controller that is used as a transparent proxy between the SDN controllers and the OpenFlow-enabled devices.

FlowVisor uses OpenFlow to control the network, the underlying physical OpenFlow-enabled devices. It also isolates a set of devices for a single controller, which is called a slice. Therefore, a controller can observe and manage its own slice, but it cannot operate with the others. Consequently, FlowVisor provides an abstraction layer.

FlowVisor takes advantage of some OpenFlow characteristics to provide bandwidth isolation and also topology isolation [37].

In conclusion, FlowVisor provides the network administrators with a software solution to slice the network into isolated portions of OpenFlow messages, controllers and devices.

Resonance

Resonance [38] is a system for enhance the security in SDN. The network devices implement dynamics access control polices thanks to the information that SDN can obtain from the data. Therefore, switches can manipulate traffic at lower layers and take actions to provide the SDN with security measures.

The architecture of resonance can be divided into three different parts. Firstly, the policy specification framework, which allows applying a set of functions to a device based on not only the security class assigned, but also the state. Secondly, we can find the monitoring part. Unlike the traditional securities systems, Resonance takes advantage of the SDN architecture to set the task of monitoring to the network itself. The monitoring is distributed-based. Network nodes send the network traffic reports to the SDN controller, which performs the analysis of the data. Finally, the last part of the Resonance architecture is the dynamic control of the network. Resonance uses the OpenFlow protocol to perform the necessary actions in order to achieve network security.

Resonance shows how SDN can provide new mechanisms to improve network security and not only network efficiency or throughput. It is interesting how SDN can provide new ways to apply policies or actions, improving the network management, not only in security, but also in other control mechanisms, like routing.

OFLOPS

OFLOPS [39] is an open software created to test the capabilities and possible existent bottlenecks between the switch and the control framework or SDN controller. It allows simulating specific usage scenarios to measure the performance of both hardware and software.

OFLOPS is a multithread software. It is written in C and it allows developers to gather information from data channels. Developers can handle events through an API to implement and measure controller functionality. OFLOPS captures data plane traffic and translate the packets to events.

1.3 Applicability and challenges of SDN

SDN improves the interoperability, the ability to manage and program the network, the performance of the network, and the security. Therefore, SDN has been used mainly in the data centers [40]. Big companies like Google have connected their data centers through an SDN [41]. This SDN provides a high performance and throughput. However, SDN can provide advantages in more scenarios than only in the data centers. Google also use SDN for WAN, interconnecting its data centers. Furthermore, they use SDN to implement BGP and provide a real time updating of the routing tables. Service Providers like Vodafone [42] offer SDN to their clients. The main reasons are security, automatization and performance. Vodafone claims that SDN is the answer to some key areas regarding networking and business such as collection, use and security of data or virtualizing processes and infrastructure [43].

In addition, the programmability of SDN allows this paradigm to adapt easier to the problems of the network, regardless its nature. This makes SDN a great option for Smart Cities [44], WSN [45], IoT [46] or Mobile Networks like 5G

[47]. Therefore, SDN is a useful paradigm to provide solutions in heterogeneous networks.

SDNs are presented as a solution to improve the resource allocation management and to provide fairness among users of multimedia streaming adaptive applications [48] [49]. SDNs present several advantages over traditional networks that place them as an excellent solution for the challenges of multimedia video streaming. The programming interface in SDN allows a better control over the network, providing an improved performance and more configuration options of the network architecture [50] [51]. Furthermore, SDN allows reconfiguring the network flows according to the needs of the users without the actual need of independently configuring all devices. As SDNs are logically centralized, the SDN controller has a global view of the state of the network supporting the dynamic optimization of data flows and the available resources [52]. Moreover, QoS can be easiliy assigned according to the flow or application selecting a priority according to QoS parameters such as bandwidth, jitter, delay, and packet loss.

Networks need a new architecture to solve the problems experienced that are related to their increase in complexity, incoherent policies, impossibility of escalation and dependence on manufacturers. SDN has a network architecture that provides, among other capabilities, the dynamism, manageability and adaptability necessary to solve the problems mentioned above. Moreover, SDN can provide support for the necessary network services. It can help in obtaining statistics of the flows that pass through the network nodes.

By using the OpenFlow standard, the SDN controller is able to communicate and operate with the network nodes, executing actions directed to improve the performance of the network, regardless the type of network. Since this type of multimedia service is used regardless the network architecture below, technologies like SDN can improve some characteristics and performance of this kind of multimedia transmission. Taking advantage of the network status and taking advantage of being able to interact and program the network can improve the QoE of the user of that service. In addition, the programming interface in SDN enables a tighter control of network resources. This can be translated into a better performance, but also into new policies or network algorithms. Since, in SDN, the applications can access to the control layer through an API, novel solutions and techniques can be applied to control the network. One of the possibilities SDN brings is to introduce AI techniques to networking. AI has been used recently in fields like Big Data [53], game production [54] or image processing [55], among others.

Despite all the commented previously, AI, thanks to the architecture of SDN, can help to manage resources and network traffic dynamically. Adapting AI to SDN could provide new policies, algorithms, protocols and ways to operate networks. Using AI to study the traffic of the network, and the statistics that can be obtained from the datapath thanks to OpenFlow, the different types of flow that are being transmitted can be discovered. Thus, traffic patterns can be obtained, which can then be applied in SDN decision-making. Combining AI techniques with SDN, adaptive behaviors are achieved in order to improve the performance of the network.

Due to the predicted change about Internet traffic previously exposed, multimedia traffic must be managed as efficiently as possible. Therefore, and knowing the capabilities of techniques such as SDN and AI, a hypothesis can be stated. This hypothesis is that combining both to operate multimedia data in heterogeneous networks can improve the performance, in terms of QoS and QoE.

1.4 Objectives and motivation

The proposal of this work is to provide a novel dynamic routing system focused on multimedia traffic. Its goal is to improve the QoS and QoE of the multimedia transmission and to be able to adapt to the problems of heterogeneous networks.

In order to achieve the goal, two different technologies are used. Firstly, SDN is the network paradigm chosen. Thanks to SDN and its architecture, the routing protocol could manage the network as a whole, taking advantage of the OpenFlow standard and the programmability of the network to make dynamical changes. The routing protocol could then take actions to guarantee a solid QoS or QoE.

On the other hand, the system uses AI to identify problems and solutions in the traffic transmissions. This, combined with the control of the SDN, would allow the system to prevent future problems and react quicker than traditional protocols.

This solution could help service providers to create an auto-managed network that guarantees QoE to the users of multimedia services. However, it also could help as a base to show that SDN has a huge potential to bring new applications and solutions to networking, not only focused on multimedia.

The main objectives of this doctoral thesis are the following:

- Introducing the use of AI for optimal route selection based on network status.
- Designing a software module to implement routing in SDN.
- Adapting an existent routing protocol to multimedia transmission using the advantages of SDN.
- Identifying actions and policies that can be used in an SDN to improve the QoS and QoE of a multimedia transmission.
- Designing of a routing protocol that uses AI to apply the actions identified before, taking into account heterogeneous networks.
- Evaluating the performance of the proposal, comparing with traditional environments and with the adapted protocol.

1.5 Precedents

Due to its novelty, SDN is a field where not many works have been turned into PhD. thesis. However, some works related to this kind of networks exists.

Laura Martín González presented in 2019 her PhD. thesis Programmable DSP-Enabled multi-adaptive optical transceivers based on OFDM technology for software defined networks. She studied the optical transmissions, addressing distance and data rate problems. She proposed programmed transceivers that achieve an efficient utilization of the optical spectrum, based on the principles of SDN. Although the field of application is not the same, by using these principles, they achieved a better performance in the transmissions due to the adaptability to the characteristics of the transmitted signals.

Angel Leonardo Valdivieso Caraguay defended his PhD. Thesis Monitoring and Discovery for Self-Organized Network Management in Virtualized and Software Defined Networks in 2017. His work was based on the architecture proposed by the SELFNET project (Self-Organized Network Management in virtualized and software defined networks). He studied the relation between the different concepts of this architecture, such as SDN, NFV, 5G, QoS and network management. He proposed in his thesis an architecture, which is focused on monitoring the network. Therefore, the heterogeneous metrics are collected and modeled into a generic data model.

Jorge Ricardo Flores Moyano presented his PhD. thesis "Contribución a las arquitecturas de virtualización de funciones de red y redes definidas por software aplicadas a las redes residenciales con gestión centrada en el usuario" in 2018. The thesis goal is to improve the performance of the residential networks. In order to achieve its goal, he presented a new architecture for managing the residential networks. This architecture was based on SDN and NFV and provided the users with the possibility of managing the network by using managing applications.

Adrián Flores de la Cruz defended his PhD. thesis "Contribución al diseño de redes definidas por software" in 2019. He presented a Key-Based routing, implemented thanks to SDN, to improve the energetic efficiency of networks. By using the OpenFlow protocol, he specifies how the SDN should address the problems of topology discovery, statistic gathering and routing. Furthermore, the routes are chosen to minimize the energy consumption, choosing paths with already-active nodes.

As regards multimedia streaming and networking, our research group has been working for a long time on video streaming and on video streaming in Ad-hoc networks.

The research group has also been involved in many theses related with the main topic of this PhD thesis, including QoS, multimedia traffic and AI.

Alejandro Cánovas Solves defended his PhD. thesis "Diseño y Desarrollo de un Sistema de Gestión Inteligente de QoE para Redes HD y Estereoscópicas IPTV" (in 29/04/2016). He proposed a management system with artificial intelligence, based on inductive prediction methods, to provide the end-user with a minimum level of QoE.

Juan Ramón Díaz Santos presented his PhD. thesis Design and Implementation of a Communication Protocol to Improve Multimedia QoS and QoE in Wireless Ad Hoc Network in 2016. It addressed the problem of multimedia delivery over multi-hop ad hoc wireless networks, especially over wireless sensor networks.

Miran Taha defended his PhD. thesis Smart Client-Server Protocol and Architecture for Adaptive Multimedia Streaming in 2018. The aim of his work was to improve and optimize the QoE. In order to do that, he characterized, designed, developed and evaluated different multimedia applications.

Furthermore, our research group has been involved in several projects regarding SDN and multimedia traffic. The last projects were "Distribucion inteligente de servicios multimedia utilizando redes cognitivas adaptativas definidas por software", where the goal was to use intelligent methods to improve the quality of multimedia services in SDN and "Red Cognitiva Definida por Software para Optimizar y Securizar Tráfico de Internet de las Cosas con Información Crítica". This last project is based on SDN to provide the IoT traffic with security methods that guarantee an optimal transmission in an SDN.

1.6 Thesis Structure

After introducing the main issues that have motivated the present Thesis, as well as a description of the main objectives that are pursued, the rest of the Thesis is organized as follows:

In chapter 2, the state of the art of SDN is presented. Related works about the study of this network paradigm are discussed. Furthermore, works regarding AI, multimedia transmission or other technologies that implies different network characteristics are also discussed.

In chapter 3, the design of the control software that is the base for the routing system implementation is presented. The details of the design, functionality and implementation are discussed.

In chapter 4, the routing protocol is detailed. The main aspects of the routing proposal such as the architecture, communication procedures and messages are discussed.

In chapter 5, the AI method used by the routing protocol is described. How the AI method should be adapted to SDN architecture for its implementation is also discussed. Finally, the solution is tested against traditional networks and other solutions.

In chapter 6, some applications in different networks are evaluated. This chapter shows some possible applications in heterogeneous networks.

Finally, chapter 7 draws the conclusion and future research. It also includes the list of publications derived from the PhD.

Chapter 2

State of the Art

In this section, the current state of the art is exposed. Some of the recent works performed by different researchers are explained and summarized. The section is organized on different subsections. The works are grouped in these sections based on their research topic.

2.1 Introduction

There are some published works that help understand the reader the concept of SDN and OpenFlow and provide some network implementations [56]. Hu et al. compiled the most important topics about SDN implementation and performed a comparison of different SDN schemes, discussing the future of this research area. Authors focused their work on how SDN separate data and control planes, making scalable networks developments based on SDN. Scott-Hayward et al. [57] highlighted the need of designing security schemes in SDN. Authors classified the security challenges attempting to which SDN layer are affected by. In addition, they proposed some solutions to these challenges and made emphasis on the importance of further work in order to achieve a secure and robust SDN environment, using the capability of being programmed and allowing a centralized network.

Kreutz et al. [58] presented a comprehensive overview of SDN and, despite its fundaments are not new, how SDN could be the new and revolutionary paradigm for Internet. Thus, they did a complete analysis of SDN infrastructure and the main challenges of SDN.

Kaur et al. [59] presented Mininet, an emulator for deploying networks on a single Virtual Machine as a method to make cheaper tests than tests over physical devices. Authors said that Mininet offered an easy use, performance accuracy and scalability for developing networks. Actually, Mininet has become a powerful tool which makes it easier to test complex networks without the need of having real hardware. In this sense in [60], Paasch et al. evaluated some scenarios for TCP multipath which could derive in huge costs in material like OpenFlow switches.

We can also find some previous works about experiments and performance tests of SDN and Mininet. De Oliveira et al. [61] explained the SDN paradigm, its elements, its purpose and its structure. They also presented Mininet and how it can help researchers to avoid the use of real and expensive hardware for networks. Authors exposed the need of making it easier to implement networks and how Mininet achieves it, with good performance and great reliability.

Moreover, Keti et al. [62] presented an evaluation of Mininet to study its limitations. The results showed that the simulation environment can generate a remarkable effect on the required time for implementing a topology.

In [63], Azizi et al. proposed a new model for measuring the delay in SDN (using Mininet), on a Multiprotocol Label Switching-Transport Profile (MPLS-TP) network. Authors proved that Mininet is not a good emulator for a stress test. However, SDN and Mininet can be used to get other measures as delay values. For instance, Gupta et al. [64] described the design, implementation and the use of fs-SDN (a simulation-based tool) to solve the problem of prototyping and accurately evaluating, at large scale, new SDN-based applications. The results enable the easy translation from virtual environment to real controller platforms like POX and NOX. Authors used Mininet in nearly identical configurations to compare it with fs-SDN.

Panwaree et al. [65] presented an evaluation of the network performance of video streaming over two kinds of OpenFlow-enabled network testbeds. They show the measurements obtained of delay and packet loss when video is streamed over Mininet and OpenvSwitch, emulating a network, installed in a PC.

2.2 SDN Controller Studies

There are several works where researchers try to analyze the performance of SDN controllers. Usually, in order to achieve this goal, they compare different SDN controllers.

In [66], Shah et al. analyzed the architecture of the most common SDN controllers (from now SDN controller are the only one that uses OpenFlow as a communication standard). In addition, they also analyzed the network performance with each of these controllers. After measuring and analyzing the architectures, the authors defined some steps to follow in order to increase the efficiency of a new controller design. With these criteria, the authors increase the throughput and enhance the latency and the scalability with the number of switches using their new designed controller.

Shalimov et al. analyzed in [67] popular open source SDN controllers. They compared NOX, POX, Beacon, Floodlight, Maestro, MuL and Ryu. The performance of the SDN controllers is tested based on scalability, throughput, reliability and security. The authors concluded that those controllers had bad scaling over the cores and were not prepared to meet increasing demands in data centers.

The performance of Ryu SDN controller is also tested in [29]. Asadollahi et al. contributed with an evaluation of scalability of Ryu controller. They introduced the architecture of Ryu and presented a simulation using Mininet to test the performance of Ryu in 6 different scenarios. They observed that Ryu is a resource demanding controller in terms of CPU and RAM, which means that, with an increment of the number of nodes, the performance can be reduced.

Other SDN controllers are also compared. In [68] and [69], the authors compared Floodlight and POX. Although both papers conclude that Floodlight outperforms POX, Bholebawa and Dalal present a better analysis in [68]. The SDN controllers are compared in different topologies: single, linear, tree and custom. These topologies are built in Mininet, being all, except the custom one, already defined by the emulator. These differences in the topologies allow the authors to test the scalability of both SDN controllers. The performance is measured in terms of Round-trip time (RTT) and throughput. The results show that Floodlight presents an improvement in RTT from 11.5% in the single topology up to 19.6% in the tree topology. Floodlight presents an improvement of 14.4% in the custom topology. As regards the throughput, Floodlight improves from 3.8% up to 8.9%.

OpenDayLight is one of the most common-used SDN controllers and there are some works measuring its performance. Khan Khattak et al. present in [70] a performance evaluation of OpenDayLight SDN controller. They present the architecture and main features of OpenDayLight. Using Cbench, a specific testing tool, the performance of the controller is measured in terms of latency and throughput in three different scenarios with different number of switches working. The authors compare OpenDayLight with Floodlight to test the performance of the SDN controller. The results show that OpenDayLight presents better latency results that Floodlight.

Rowshanrad et al. also compared OpenDayLight with Floodlight in [71]. The SDN controllers were compared based on their delay and loos performance, in different topologies and network loads. The authors shown a comparison of the features and characteristics of both controllers, such as programming languages, REST API, documentation and so on. The test was performed using the Mininet emulator and the network topologies tested were single, linear and tree. Results show that OpenDayLight presents a better performance in networks with low load. However, Floodlight shows a better performance in latency and loss for linear and tree networks with heavy traffic. Therefore, the SDN controller should be chosen depending on the network's features.

SDN controllers can be also studied from a security-based point of view. In [72], Arbettu et al. ran a security analysis of some controllers like OpenDayLight and Ryu. They use the STRIDE [73] framework to create threat and test the performance of the SDN controllers in terms of security. The threats used and tested are: spoofing, tampering, repudiation, information disclosure, denial of serviced and elevation of privileges. The controllers were tested so that authors can tested if the controller is affected or not by the threat or if it has mitigation mechanisms. The results show that OpenDayLight is the most secure SDN controller.

2.3 Improving QoS or QoE

In the last years, due to the increase of network traffic, many works have been proposed to classify multimedia traffic in order to guarantee QoS and QoE. In [74], authors proposed a multimedia traffic classification model based on patterns by using video streaming and network characteristics as input parameters and video quality evaluation measures as output. They use NQI, VQM, Structural Similarity Index (SSIM), and Peak Signal-to-Noise Ratio (PSNR) metrics to conform traffic patterns. Different learning methods are studied to find out which of them provides the best result. These methods are based on NNs, vector support machines, statistics and the nearest neighbors. In [75], is presented an experimental comparison of the performance of SNR, SSIM, NQI and VQM. Depending on several parameters, the best performance can be obtained from a different method, but in general, the best results are given by VQM.

In [76], it is developed a novel framework called DeepQoE to predict video QoE. The end-to-end framework first uses a combination of deep learning (DL) techniques (e.g., word embedding) to extract generalized features. Next, these features are combined and fed into a NN for representation learning. Such representations serve as inputs for classification or regression tasks.

Authors, in [77], try to solve the bandwidth allocation problem in cloud computing data center networks using SDN paradigm. This method makes use of machine learning techniques to classify the incoming traffic flows in real-time while ensuring game flows are prioritized over others. In [78], authors use machine learning techniques to try to demonstrate how QoS metrics can be exploited to accurately estimate and predict key QoE factors. They show that context information on network congestion and basic characteristics on video streams further improves predictions.

Some authors evaluate the QoE for IoT applications, with particular attention to multimedia traffic, like in [79]. They introduce the concept of Multimedia IoT (MIoT) and define a layered QoE model aimed at evaluating and combining the contributions of each influence factor to estimate the overall QoE in MIoT applications. In [80], authors try to predict video QoE based on information directly extracted from the network packets using a DL model. The QoE detector is based on a binary classifier (good or bad quality) and uses seven common classes of anomalies. The proposed classifier is based on a combination of a Convolutional Neural Network (CNN), recurrent neural network, and Gaussian process classifier. A deep subjective study of video quality has been performed in [81]. It is also evaluated the performance of several state-of-the-art, publicly available full-reference video quality assessment algorithms.

Authors, in [82], study the adaptive multimedia traffic control mechanism leveraging Deep Reinforcement Learning (DRL) that combines DL with reinforcement learning (RL), which learns from rewards by trial-and-error. This mechanism is able to control multimedia traffic directly from experience without referring to a mathematical model.

Authors, in [83], analyze several researches about application of Machine Learning (ML) techniques to IP traffic classification. They also discuss a number of key requirements for the employment of ML-based traffic classifiers in operational IP networks. In [84], authors employ three supervised machine learning algorithms, Bayesian Networks, Decision Trees and Multilayer Perceptrons for the flow-based classification of six different types of Internet traffic. Their experiments show that ML algorithms such as Bayesian Networks and Decision Trees are suitable for Internet traffic flow classification at a high speed.

In [85], authors apply a Naïve Bayes estimator to categorize traffic by application, showing its high level of accuracy. In this approach, they use samples of well-known traffic to allow the categorization. A fully automated Packet Payload Content (PPC) based network traffic classification system is presented in [86]. The proposed system learns new application signatures in the network where classification is desired. Furthermore, that system adapts the signatures as the application traffic changes.

In [87], it is analyzed how different factors contribute to the QoE in the context of video streaming delivery over cloud networks. The authors describe different methods that are often used to collect QoE datasets in the form of a Mean Opinion Score (MOS). Machine Learning methods are then used to classify a preliminary QoE, and six classifiers are evaluated and determined. Also, in [88], the authors present a user QoE prediction algorithm which extracts features based on user traffic pattern parameters such as bitrate, resolution, frame rate, etc. They use three different feature selection algorithms and six different classifiers in order to optimize the features set and the corresponding ML algorithms.

L. Huixian et al. develop a traffic-analysis method using an unsupervised ML technique, where flows are automatically classified by exploiting the different statistics characteristics of flow [89]. Authors, in [90], develop a machine learning based traffic classification based on statistical properties while optionally decoupling flow classification and treatment. This decoupled architecture allows centralized traffic classifiers to control traffic filtering and shaping by diversely located, low-performance network devices.

There are authors who study multimedia transmission based on SDN technology. In [91], Taha et al. present an algorithm for the management of video transmission performance based on SDN. Their algorithm provides a stable video because it distributes bandwidth equally among clients. Besides, they perform tests and present some results that confirm the validity of their algorithm.

O. Awobuluyi et al. presented in [92] an SDN controller for scalable video encoded over H.265. The information on the scalability of the multimedia streams and network capabilities was known by the controller that made the decisions to ensure QoS and QoE over 5G networks. The proposed architecture was comprised of a Video Quality Assurance Manager (VQAM) that gathered information on the topology and video metrics to decide the routing and video adaptation. It included an SDN Video Quality Orchestrator (SDN-VQO) as well as to manage multimedia flows and to ensure fairness. J. Castillo et al. proposed in [93] a modification of the Entity Title Architecture (ETArch) employing OpenFlow and the QoS control method from SMART (Support of Mobile Sessions with High Transport Network Resource Demand). It provides coordination over dynamic and admission control of super-dimensioned resources in order to provide QoS and QoE to multimedia flows. Tests were performed on a real testbed obtaining the admission of all the sessions with a network saturation up to 170%, reaching a 27% of SSIM improvements over legacy ETArch and reducing the signaling load of legacy ETArch in a 251%.

An NFV architecture for managing and controlling multimedia applications over 5G with SDN and considering QoE requirements was introduced by A. A. Barakabitze et al. in [94]. NVF and SDN are utilized to reach the Key Performance Requirements/Indicators (KPR/I) of 5G. Different Virtual Machines (VMs) were utilized to address different network requirements of resources of the multimedia applications. The QoE was assessed utilizing the QoE-sdnFlow manager and the QoE-sdnFlow Monitor. The type of multimedia applications was considered to provide the specific resources required by the user.

Lastly, T.-N. Lin et al. presented in [95] a meter-based multicast method to provide end-to-end QoS for multimedia services called OpenE2EQoS. The designed algorithm utilizes a learning mechanism to allocate the available bandwidth for the multimedia flows in order to guarantee QoS. The congestion problem in links is addressed by forwarding low priority packets among N different routes. Therefore, the problem of forwarding the multimedia flows to other routes is prevented. Tests were performed in a real SDN environment where the effectiveness of their proposal is demonstrated.

Optimization has been utilized as well as a tool for increasing QoS and QoE. A. Kassler et al. presented in [96] an SDN-based QoE-driven centralized multi-user path optimization for multimedia services. It maximized QoE considering link capacities, delay, network topology and service utility functions. The user was able to select video quality over audio quality and vice versa. Therefore, video quality may be degraded while audio quality is maintained high enough when the resource availability of the network decreases. OpenFlow was utilized to determine the paths according to the results of the optimization. A new OpenFlow controller called OpenQoS was introduced by H. E. Egilmez et al. in [97]. OpenQoS was intended for multimedia delivery and QoS support. The paths of the multimedia flows are optimized so as to ensure QoS. The performance of the proposed controller was compared to the exiting HTTP-based multimedia adaptive streaming. Results showed that OpenQoS outperformed HTTP-based multimedia adaptive streaming, guaranteeing seamless video delivery in UDP and full video quality in TCP.

Finally, E. Grigoriou et al. presented in [98] an SDN-based resource management mechanism for multimedia services that optimizes the QoE of end users. In order to do so, the service was divided into different tasks assigned to the neighbor nodes taking into consideration the final quality. Authors stressed the enhancement of QoE management, resource allocation and video quality that SDN/NFV provided. Tests were performed with the OpenDaylight controller and the Mininet network emulator. Results showed an improvement in video quality.

Noghaniet et al. [99] introduced a framework based on SDN that could enable the network controller to deploy IP multicast between source and subscribers. The network controller was also able to control the distributed set of sources where multiple description coded (MDC) video content is available by using a simple northbound interface. Due to this SDN-based streaming multicast framework for medium and heavy workload, the PSNR of the received video is increasing considerably. Authors noticed that the received video, which had a very poor quality before, was having a significant increase in the quality of video now.

Nam et al. [100] proposed a mechanism to solve the congestion problem and improve the video QoE. Authors tried to develop an SDN based application to improve the quality of video that can monitor conditions of network in real time streaming, and change routing paths dynamically by multi-protocol label switching (MPLS).

Egilmezet et al. [101] give a unique design of an Openflow controller for multimedia delivery over SDN with end to end QoS support. The authors tried to optimize routes of multimedia flows dynamically. After experiments over real test network, the authors found better results than HTTP based multibitrate adaptive streaming. They ensured that OpenQoS can guarantee the video delivery with little or no video artifacts experienced by the end-users. In another publication, Egilmezet et al. [102] gave new distributed control plane architectures for multimedia delivery over large-scale, multi-operator SDN. The extensions included in the design of the architecture were: (a) to acquire network topology and the state information by topology aggregation and link summarization, (b) to propose an optimized framework for flowing based end to end over multi-domain networks, and (c) two distributed control plane designs by addressing the messaging between controllers for scalable and secure routing between two domains. By applying these extensions on layered video streaming, authors obtained a better quality of received video, reduced cost and memory overhead. This architecture was effectively scalable for large networks.

Kassleret et al. [96] tried to negotiate the service and parameter for network communication between end users and to assign multimedia delivery paths in the network according to prefixed service configuration. The idea behind this system was to centralize multi-user optimization of path assignments, which provides the better QoE by considering network topology, link capacities, delay and account service utility. Due to optimization, the system was able to use Openflow to set up forwarding paths in the network.

Mehrdad Hosseinnejad Yami et al. demonstrate the importance of the game-state as a criterion for game requirements [103]. They used and designed an SDN architecture to make easier the communication between game and server. They also propose a resource allocation algorithm that considers measurements such as delay, bandwidth and game-state information. Finally, the system is implemented and evaluated in a simulated environment, simulating the QoE via extensive subjective tests. Results show the superiority of the proposal over existing methods “on average 1.26, 1.18, 1.21, and 1.12 increase in QoE, responsiveness, controllability, and immediate feedback, respectively, in a scale of 0-5”.

In this sense, E. Liotou et al. [104] presented the concept of an SDN-based QoE-Service as a manner of ensuring an acceptable QoE level for on-demand services or premium users of over-the-top (OTT) applications. To achieve this goal, authors explain that an SDN-based QoE-Service is required to be able to monitor the network parameters by the SDN controller and the mapping of these parameters to QoE values inside the QoE Server.

A. Martin et al. [105] presented a network resource allocator system which enables autonomous network management aware of QoE. As we can see, the system is able to predict demand to foresee the amount of network resources to be allocated. In addition, the system dynamically provides the network topology in a proactive way while maintaining the QoS ranges. The system works as a collaborative network to process signals from multiple network nodes and end-to-end QoS and QoE metrics. According to the results, this kind of system can be optimal for live and on-demand dynamic adaptive streaming over HTTP while maintaining the high efficiency video coding services. Authors concluded that the system is able to scale the network topology and to address the level of resource efficiency required by this kind of services.

Finally, we should consider that HTTP adaptive streaming (HAS) is being consolidated as a standard tool for online video distribution. While existing network infrastructures usually prioritize the fast content forwarding and not the quality of the delivered content, the future network tends to manage and adapt the content format to user-level cognitive factors for effectively and intelligently allocating limited network resources. M. Mu [106] proposed a software defined cognitive networking (SDCN) aimed to incorporate new developments inhuman cognition, media technology and communication networks to ensure the user experience, user-level fairness and network efficiency of online adaptive media. To achieve this goal, the proposal is based on an optimized user-centric resource model, which oversees paired with tailored SDNs functions to form a service for QoE-aware in-network monitoring and resource allocation.

Furthermore, Megyesi et al. [107] introduced a novel mechanism for measuring available bandwidth in SDN networks. They built an application over the Network Operating System (NOS) which was able to track the network topology and the bandwidth utilization over the network links, and thus the mechanism was able to calculate the available bandwidth between any two points in the network. Authors validated their method using Mininet network emulation environment. We have seen some deployments that use SDN based testbeds for multimedia streaming.

Noghani et al. [108] proposed an SDN-based IP multicast framework in order to control a set of video sources and the impact of SDN on QoE. This framework significantly increased the PNSR of the video, so the user received a good-quality video from a video almost impossible to see. Another practical case is shown by Michael Jarschel et al. [109]. They used a video streaming service like Youtube, in order to evaluate the performance over SDN see how OpenFlow can help us to improve the QoS in video delivery applications.

SDN networks can also be mixed with IP networks. Salsano et al. [110] described the design and implementation of an Open Source Hybrid IP/SDN (OSHI) node. They provided a set of open source tools that allowed facilitating the design of hybrid IP/SDN experimental networks. Their work shows the deployment on Mininet and on distributed SDN, and their test bench. In this way, it is possible to evaluate the costs related with the SDN integration on the Internet.

2.4 SDN Applications

There is a lot of previous work dealing with the study of multimedia transmission, video surveillance, IoT, AI and SDN. Even many of the previous works interact with different technologies. In this subsection, applications of several technologies, usually combined with SDN, are presented.

There are works done by other authors that relate SDN with IoT. Huang et al., in [111], expect that as the SDN networks proliferate, the collection of information and the updating of the network topology and the control of QoS will be facilitated in the IoT environment. Quin et al., in [112], extend the Multinetwork Architecture Information Architecture (MINA), achieving different levels of quality for different IoT tasks so that an original SDN IoT controller supports commands to differentiate flows and tasks. They have applied a prototype to an IoT scenario. The performance results indicate that IoT networks can be exploited more efficiently. In [113], Omnes et al. present a new multilayer IoT architecture, which includes SDN and NFV, based on network and IT resources.

Some authors, such as Bizanis et al. in [114], study the state of the art when applying SDN and Network Virtualization (NV) to IoT. They describe the implementation of IoT in both technologies and finally review IoT architectures enabled by SDN-NV together with implementations in real life. Other authors, such as [115] and [116], describe SDN presenting their first fields of application, and analyze the possibility of using it in IoT applications.

There are authors who study different areas of IoT applicability. For example, Pal, in [117], establishes six areas, called SPACES, which are of interest to companies and organizations when implementing IoT. The areas he presents in his study are: scalability, privacy, affordability, context awareness, ease-of-development, and security. Other authors, such as [118] and [119], propose to use IoT devices and sensors in the field of e-Health or Ambient Assisted Living (AAL). Both papers discuss the possibility of obtaining information through these devices to improve the personalization of medical treatment, facilitate the practice of medicine, and reduce costs. Tan et al., in [120], present the structure of IoT; they also propose IoT architecture and design an application model. Gubbia et al., in [121], present a vision of the implementation of IoT in the Cloud. They study key technologies and application domains, to subsequently implement a solution based on both public and private clouds. This implementation is done within a framework that allows Cloud scalability, and that provides capacity for IoT.

Other authors study in the scope of video surveillance. Authors, such as Ajiboye et al. , in [122], propose a new hierarchical architecture called Fused Video Surveillance Architecture (FVSA). Privately-owned video surveillance systems can increase efficiency in public safety. In their proposal, they define a network adapted to intelligent services of video surveillance, which allows communicating with other compatible systems in IoT. Lloret et al., in [123], present the study of the implementation of a video surveillance system in rural environments. They study codec selection, design and coverage problems, and finally show the results obtained in a public deployment.

In [124], Yang et al. have proposed a novel time-aware software defined networking (TaSDN) architecture for OpenFlow-based data center optical networks, by introducing a time-aware service scheduling (TaSS) strategy. The strategy can arrange and accommodate the applications with required QoS considering the time factor, and it enhance the responsiveness to quickly provide for data center demand.

Dramitinoset et al. [125] have discussed different aspects of video delivery over next generation cellular networks, which includes the SDN and cloud computing. The authors have been focused on next generation cellular networks which employ SDN in core due to increased demands of video streaming commercially. In our paper we are trying to explore the performance of multimedia delivery over SDN as compared to real test networks in terms of some important parameters.

P. Gallo et al. [126] showed a proposed framework to manage current home networks composed by network segments with different requirements and features. Authors analyzed the problems and flaws of current approaches that make difficult to provide the interoperability of these technologies. Finally, authors presented the vision of an SDN multi-technology network architecture called SDN@home and showed how a future home gateway which would be the SDN controller can directly and dynamically program network devices. In addition, authors defined a new type of flexibility enabled by SDN@home since wireless protocols and features are tending to no being longer tied to a specific technology although they could be used by general-purpose wireless SDN devices.

2.5 Routing in SDN

There are also authors who propose the use of SDN to obtain greater efficiency in routing, not only using traditional protocols such as OSPF, IS-IS [127] or BGP [128], but also some hybrid solutions.

The authors Caria et al., in [129], present a hybrid operation mode SDN/OSPF. In their proposal, they use SDN nodes to divide the OSPF domain into subdomains. Within each subdomain, the routing remains stable. The SDN nodes are located at the borders of the subdomains, and they are responsible for adjusting the routing updates. In this way routes between subdomains can be optimized. According to their simulation results, it is not necessary to deploy SDN on all nodes of the network. Similarly, Rothenberg et al., in [130], propose a controller-centric hybrid networking model and present the design of the Route-Flow Control Platform (RFCP) along the prototype implementation of an AS-wide abstract BGP routing service.

Also, there are authors who propose the use of SDN in ad hoc vehicular networks (VANET). Ming et al., in [131], indicate that in order to efficiently send information in VANET, a protocol that has a short delivery delay time and where low routing overhead is required. They propose a routing protocol based on SDN in which a central controller collects all the information of the rest of the controllers and calculates the optimal routes, based on all the information collected in other points of the network.

H. Nam et al. presented in [100] an SDN application to monitor the network and assign routing paths to video content utilizing MPLS in order to ensure QoE. In this case, the routing solution was focused on improving QoS and QoE. The Junos Space SDN platform was utilized to implement the proposal. Tests were performed in different scenarios employing a simulation tool. Results showed an improvement of the viewing experience of 55.9% in peak hours.

Abbas, M.T et al. [132] proposed a routing protocol for internet of vehicles. For that, they introduced an SDN-based network architecture. They claim that SDN enables the routing protocol to handle highly dynamic networks in an abstract way by dividing the data plane from the control plane. Then, road-aware routing strategy is introduced. After that, a novel mathematical model is estimated whose goal is assists the primary controller to find the shortest path. The results show the performance of the proposal protocol in terms of availability with limited routing overhead.

Á L. Valdivieso et al. proposed in [133] an SDN-based framework to optimize multimedia routing. The framework utilizes NFV and OpenFlow in order to test the routing algorithms. The effectiveness of the framework is determined by the QoS routing algorithm and network performance modules. The parameters analyzed were PSNR, SSIM and MOS from both best-effort and optimized routing. Results showed an improvement in terms of QoS of the proposed framework compared to the best-effort approach.

M. Amiri et al. [134] presented a Linear Programing (LP) optimization-based method for efficiently assign game servers for gaming sessions. The system is able to choose the best communication path within a cloud gaming data center. The proposed optimization model takes into account the sort of requested games, the available server loads, and the current path delays. The results showed that the proposed LP model is able to minimize the average delay of all players within a data center by 9.6%, and it outperforms the existing server-centric and network-centric models.

2.6 AI Routing

Taking into account the traffic that is generated in the current networks, we are faced with the need to improve and adjust the classic routing protocols. In this way, data transmissions must be optimized especially transmissions that require special treatment to be able to meet specific requirements due to their nature. This section presents some of the most interesting proposals where authors have performed different experimental tests with network protocols and have tried to improve them by modifying the original operation, introducing AI techniques.

The performance of Interior Gateway Protocols has been evaluated in several works. In general, all authors recommend the use of the OSPF protocol as the Interior Gateway Protocol in the networks. Authors such as Sendra et al., in [135], have studied the most used Interior Gateway Protocols. In their conclusions, they say that the OSPF protocol must be chosen by network administrators when there is restriction in the bandwidth of the network they manage. Also, other authors such as Rakheja et al., in [136], have conducted studies comparing the performance of the RIP, OSPF, IGRP and EIGRP protocols. In their conclusions, they assert that the OSPF protocol is the one with the best overall performance.

During the last years, there have been multiple works in which their authors show improvements for the routing protocols based on AI.

Barbancho et al., in [137], present the SIR (Sensor Intelligence Routing) algorithm. This new routing algorithm is aimed at achieving a better QoS. It applies to an artificial neural network based on Kohonen's self-organized feature map. Moreover, Barbancho et al., in [138], compare the performance of their SIR routing algorithm with directed diffusion and Energy-Aware Routing. Finally, they affirm that the inclusion of AI techniques in WSN improves the network performance.

Many authors have studied the use of Swarm Intelligence (SI) to solve the problem of adaptive routing in telecommunication networks. Some of these authors use SI applying intelligence models based on biological swarm (ant colony optimization, particle swarm optimization, swarm robotics, and other swarm intelligence algorithms) for solving problems in the real world of sensor networks.

Arabshahi et al., in [139], study solutions to the problem of routing in wireless networks using SI as a key. They seek to maintain the desired QoS, checking bottlenecks and looking for an adaptive network.

Gunes et al., in [140], present a new protocol called Ant-Colony-Based Routing Algorithm (ARA) that is highly adaptive, efficient and scalable. Their protocol is based on SI, focused on the ant colony based meta-heuristic. The main goal in the design of the protocol was to reduce the overhead caused by the routing protocol.

As actual networks are increasingly large, dynamic and heterogeneous, Ducatelle et al., in [141], indicate that, for their control and management, they require algorithms and novel protocols that are completely distributed. Moreover, they should be at the same time adaptable, robust and scalable. These protocols also allow the network to behave as an autonomous and self-organizing system. In their work, they review several SI applications, considering routing algorithms in wired and wireless networks, while establishing principles to apply SI to the design of routing algorithms. They also indicate that research fields such as gossip/epidemics algorithms are closely related to SI.

Rajagopalan et al., in [142], present a routing protocol called Ad hoc Networking with Swarm Intelligence (ANSI). Their protocol, through self-organized SI mechanisms, makes better sending decisions than traditional MANET protocols, as they gather more information. In their study, they perform simulations comparing the ANSI and AODV (Ad hoc On Demand Distance Vector) protocols [143]. They verified that ANSI provides better performance results and fewer route errors.

Authors such as Zungeru et al., in [144], show a comparative study of classic routing protocols regarding the use of SI in WSN. They also present the results of a simulation of different protocols in MATLAB to serve as a reference in the future for other researchers.

One of the preferred techniques to carry out routing in WSN, maintaining its maximum useful life, is clustering. Karaboga et al., in [145], present a clustering mechanism based on the artificial bee colony algorithm, to prolong the life of the network. They compare the proposed algorithm with LEACH-based protocols. According to the results obtained, Clustering, based on the artificial bee colony algorithm, can be applied to WSN routing protocols.

An interesting approach, that researchers are currently investigating is the application of reinforced learning in routing protocols [146]. For example, R. Desai and B. P. Patil [147] presented a new learning strategy method to select the best route in ad hoc networks. The proposal is based on the cooperative RL problem. To develop it, authors have based their proposal in models of social insect behavior, such as swarm intelligence. They also performed the analysis and test bench of their proposal by comparing with existing routing protocols. Results show that performance parameters such as packet delivery ratio as a function of packet rate or number of nodes in the network are significantly better when reinforced learning is used.

J. Solanki and A. Chauhan [148] presented some modifications over the AODV protocol. Authors' aim was to improve route error tolerance mechanism of this widely used routing protocol. These modifications were related to parameters as End-to-End delay, Packet delivery ratio, Traffic overhead and Power Consumptions. The proposed scheme was based on the fact that transmission started from the closest neighbor node when a link failed in the middle of the transmission. The simulation results show the proposed scheme presents very important reductions in delay and it improves the packet delivery ratio comparing with the traditional routing scheme of AODV. Results also show an interesting improvement in the routing overhead by reducing the frequency of route discovery process.

2.7 Conclusion

In this chapter, the state of the art has been presented. It described the works regarding:

- Studies about the performance of the different SDN controllers.
- Previous works presented about QoS or QoE enhancements technics and protocols.
- Contributions related to several SDN applications, using their advantages to provide services.
- Proposals describing how routing could be solved in SDN.
- Researches discussing the introduction of AI methods in routing.

With all these works analyzed and knowing the current state of the SDN related works, the proposed solution of this dissertation can be discussed.

Chapter 3

Design and Implementation of a Custom SDN Controller

In order to design, create, implement and test a network protocol, it is a key factor how much control over the network is possible to have. Open-source controller like Floodlight, POX or OpenDayLight allows a limited control, which varies depending on their architecture. A simple SDN controller was developed from the scratch to fulfill the implementation and testing requirements. The controller is fully developed in Java. In this chapter, the design and implementation of the custom SDN controller is presented. However, before explaining the controller, an SDN background is discussed to fully understand the concepts managed from this point on.

3.1 SDN Background

In this section, the technical concepts of SDN needed to understand the proposal are detailed. First, the concept of routing in SDN is commented. Then, the OpenFlow protocol is detailed. Finally, virtualization in SDN is addressed.

3.1.1 Routing fundamentals

In traditional networks, routing protocols work commonly in a distributed among the network nodes. Network nodes exchange information to update the cost of reaching certain nodes.

Depending on what information is exchanged, and how it is exchanged, the algorithms can be divided into two groups. Distance-vector algorithms use the Bellman-Ford algorithm to calculate the cost. Each node sends information about changes in topology and distances to its neighbors. RIP [149] is the most-commonly-used routing algorithm that is based on distance-vector.

On the other hand, link state algorithms are based on Dijkstra algorithm. The nodes first gather the information about their neighbors and, after that, they cast the information through the whole network to create a map of the topology, with the cost to reach each node of the network.

OSPF [150] is one of the network protocols, of hierarchical interior routing, more used in the current networks. It uses the status of their links to calculate the optimal route to the destination, and employs a metric called cost, whose value is determined by the division of a reference bandwidth between the actual bandwidth of the interface.

Regardless the algorithm behind the routing protocol, all the protocols used for routing are based on a metric which allows sorting paths from the best option to the worst. After that, each node (router) analyses the destination IP address and all the packets that match that destination IP address will be forwarded through the same interface or path, which is determined by the metric.

SDN changes this focus on the IP address and on the packet as a unit of forwarded data. In SDN, the data is grouped by flows. The OpenFlow switches store the flow information in flow tables. The flow table contains a set of entries, known as flow entries, several counters used for statistics and a set of actions to apply to matching packets.

These flows represent a set of packets with similar characteristics. The characteristics of the flows allow the controller to perform the same action for every packet that matches these characteristics. Table. 3.1 shows the fields from packets used to match against flow entries defined by the OpenFlow 1.0 standard [151].

Table 3.1: Fields used to match flow entries.

Field	Description
In port	The input port of a packet
DL SRC	The MAC source address of a packet
DL DST	The MAC destination address of a packet
DL VLAN	The input VLAN id of a packet
DL VLAN PCP	The input VLAN priority of a packet
DL Type	The Ethernet type of a packet
NW TOS	The IP Type of Service of a packet
NW Proto	The IP protocol of a packet
NW SRC	The IP source address of a packet
NW DST	The IP destination address of a packet
TP SRC	The TCP/UDP source port of a packet
TP DST	The TCP/UDP destination port of a packet

All the packets that are processed by a switch in an SDN are compared against the flow table. There are different levels of priority, so firstly the highest priority entries are checked and after that the others. If a matching entry is found, the actions set in the entry are performed on the packet. If no match is found, the packet is returned to the controller, using an OpenFlow Packet-In message, through the secure channel.

Unlike the routing in traditional networks, managing flows with characteristics from different network layers allows the routing protocol to be more flexible. Consequently, a set of actions can be specified for a concrete flow. That means a set of packets can be treated differently from the others, regardless the destination network address.

3.1.2 OpenFlow Communication and Messages

The OpenFlow standard is the communication protocol used between the SDN controller and the nodes of an SDN. These nodes, which use the OpenFlow standard as their communication protocol, may be called OpenFlow-enabled in the literature.

The communication protocol is established through a secure channel. This secure channel is the interface that connects the controller to an OpenFlow switch. The SDN controller can, through this interface, manage the switch,

Table 3.2: Structure of the OpenFlow header.

Field	Length (bits)	Description
Version	**8**	The version of the OpenFlow standard used in the message
Type	**8**	The type of message
Length	**16**	Length of the message in bytes
XID	**32**	Identifier

configuring its flow tables, receiving statistics and events from the switch and also receiving packets from the switch or send packets out the switch.

Although this interface is implementation-specific, all the messages between the SDN controller and the nodes must follow the OpenFlow standard. The format of these messages is described in the next subsection.

OpenFlow Messages

The structure of the OpenFlow messages is described in this subsection.

OpenFlow Header

All the OpenFlow messages are introduced by an OpenFlow header. This header, which is a Type-length-value (TLV) header, helps the controller to know how it must process the following data. Its first field is the version of the OpenFlow protocol used in this message. Then, the type of message is indicated. The third field is the total length of the message. Finally, an id of the transaction is sent to facilitate pairing. Table. 3.2 shows the fields and the length of the OpenFlow header.

The next messages are all introduced by an OpenFlow header. The type value determines which message is sent. Each message has its own value for the type field of the header. Therefore, when, in this work, it is said that the type field is set to a value of a message, this value is the one that represents that message for the type field of the header. Moreover, in the next messages, the OpenFlow header is not discussed because there is the first field of every message.

HELLO Message

The hello message starts a communication, and it is used in the handshake process, which is explained in the next subsection. In OpenFlow, the hello message is an OpenFlow Header whose type field is set to the Hello value.

Features Request Message

The features request message is another message used in the handshake process. It allows the controller to ask for the features of a switch in the SDN. Like the hello message, it does not contain any body. The features request message is an OpenFlow header with the type field set to the value of features request.

Echo Request Message

The echo request message is used to check the status of an established connection. It consists of an Openflow header plus an optional timestamp to check latency.

Echo Reply Message

The echo reply message not only is composed by the Openflow header, but also it must leave unmodified the data field of the echo request message. That means, the value of the optional timestamp has to be the same as in the request message.

Features Reply Message

The features reply message is the one used as a reply of the features request message. It is sent by the switches to the controller. Table. 3.3 lists all the fields sent by the datapath. In the message, the datapath identifier is sent, which the MAC address with an implementer-defined ID appended in the upper bits. After the identifier, some characteristics of the switch are detailed. The number of packets the switch can buffer at once, the number of flow tables supported, a list of “capabilities”, defined in Table. 3.4, and the actions the controller can set in the flow table. These actions are explained in the flow mod message subsection. Finally, the switch sends a description of its physical ports, following the structure described in Table. 3.5. The curr, advertised, supported and peer fields of this structure are bitmaps. These bitmaps indicate

Table 3.3: Structure of the OpenFlow features reply message.

Field	Length (bits)	Description
Datapath ID	**64**	The identifier of the datapath. It contains the MAC address
N Buffers	**32**	The maximum number of packets that can be buffered at once
N Tables	**8**	Number of tables supported by the datapath
Pad	**24**	Align to 64 bits
Capabilities	**32**	Capabilities of the datapath
Actions	**32**	Actions supported by the datapath
Ports	**48 each port**	Description of the physical ports of the datapath

the bandwidth of the link, whether it is half or full duplex, copper or fiber and if it allows auto negotiation and pause. Depending on the number of ports, the size of the message can vary.

Get Configuration Request Message

The SDN controller can ask for the current configuration of a specific datapath by sending this message. Since it is a simple request, the message is composed only of the OpenFlow header with the adequate type value set.

Get Configuration Reply Message

When the SDN controller requests the configuration of a switch, a get configuration reply message is returned. The fields of this reply are shown in Table. 3.6. The first field is a flag that represents how IP fragments should be handled. The possible values are described in Table. 3.7. The second field is the maximum number of bytes that the datapath can send to the controller when a packet does not match any flow entry.

Table 3.4: Description of the capabilities bitmap.

Field	Bit number)	Description
Flow Stats	**0**	Allow statistics per flow
Table Stats	**1**	Allow statistics per table
Port Stats	**2**	Allow statistics per port
STP	**3**	It is compatible with the 802.id Spanning Tree protocol
Reserved	**4**	Reserved, must be zero
IP Reassemble	**5**	The switch can reassemble IP fragments
Queue Stats	**6**	Allows statistics per queue
ARP Match IP	**7**	Match IP addresses in the ARP packets

Table 3.5: Physical port structure.

Field	Length (bits)	Description
Port number	**16**	Number of the port
Hardware Address	**48**	MAC address of the port
Config	**32**	Current configuration of the port
State	**32**	Current state of the port
Curr	**32**	Current features
Advertised	**32**	Features being advertised by the port
Supported	**32**	Features supported by the port
Peer	**32**	Features advertised by peer

Table 3.6: Structure of configuration message.

Field	Length (bits)	Description
Flags	**16**	Flags about the configuration of the IP fragments management
Miss Send Len	**16**	Maximum number of bytes of a packet sent to the controller

Table 3.7: Possible values of the flag field.

Possible Option	Value	Description
Normal	**0**	No special handling for the IP fragments
Drop	**1**	Drop IP fragments
Reassemble	**2**	Reassemble the IP fragments
Mask	**3**	Bitmask of flags dealing with fragmentation.

Set Configuration Message

The set configuration message is the message used by the SDN controller to modify the current configuration of a switch. It is composed of the same fields that the get configuration reply message. Therefore, the fields of the message are shown in Table. 3.6.

Flow Modification Message

The flow modification message, also known as flow mod or OFP_FLOW_MOD is the most important Openflow message in this work. When an SDN controller sends this message to a datapath, it modifies zero or more flow entries of a table. The fields of this message are shown in Table. 3.8. The most important fields are the command, the timers, the actions and the match, which indicates which flows are affected by the message when editing a flow entry. However, if a new flow is added it indicates which packets will match against the flow entry. Therefore, defining properly the match field is extremely important. The fields of the match structure are the ones defined in Table. 3.1. The command indicates if a flow will be added or removed or modified. The timers indicate when the flow entry expires. Finally, the action indicates what the datapath should do when a packet matches the flow entry. The possible actions to be

Table 3.8: Structure of the flow modification message.

Field	Length (bits)	Description
Match	**320**	Fields to match and their values
Cookie	**64**	Controller-issued identifier
Command	**16**	Command of the Flow Mod message
Idle timeout	**16**	Idle time before discarding the flow entry
Hard timeout	**16**	Maximum time before discarding the flow entry
Priority	**16**	Priority level of the flow entry
Buffer ID	**32**	The flow entry will be applied to the buffer set. The "-1" value means no buffer
Out Port	**16**	When deleting entries, the out port must be included
Flags	**16**	Flags to activate extra behaviors such as force the switch to send a message after a flow expires
Actions	**It depends on the action**	Actions to be performed after a match

performed are listed in Table. 3.9. This message will be the base of the routing protocol since it will be used to set the rules for each flow.

Port Modification Message

The port modification message allows the controller to modify the behavior of a physical port. The fields used in the message are described in Table. 3.10. The fields used in the message are similar to the ones in the physical port description structure (Table. 3.5).

Table 3.9: Possible actions to be performed by a datapath after a match.

Action	Description
Output	The datapath forwards the matched packets through a specified port
Set VLAN ID	Set the 802.1q VLAN ID to the matched packets
Set VLAN PCP	Set the 802.1q priority to the matched packets
Strip VLAN	Strip the 802.1q header of a matched packet
Set DL SRC	Set the specified MAC source address to the matched packets
Set DL DST	Set the specified MAC destination address to the matched packets
Set NW SRC	Set the specified IP source address to the matched packets
Set NW DST	Set the specified IP destination address to the matched packets
Set NW TOS	Set the specified IP Type of Service value to the matched packets
Set TP SRC	Set the specified TCP/UDP source port to the matched packets
Set TP DST	Set the specified TCP/UDP destination port to the matched packets
Enqueue	The datapath forwards the matched packets through a specified queue attached to a port
Vendor	Used for vendor's extensions of the protocol

Table 3.10: Port modification message.

Field	Length (bits)	Description
Port number	**16**	Number of the port
Hardware Address	**48**	MAC address of the port
Config	**32**	Current configuration of the port
Mask	**32**	Indicates which features of the configuration must be set
Advertised	**32**	Features being advertised by the port
Pad	**32**	Pad to 64 bits

Table 3.11: Possible values of the type field on a statistic message.

Type	Description
Description	General description of a datapath
Flow	Statistics regarding a concrete flow
Aggregate	Statistics about a set of flows
Table	Statistics of a flow table of a datapath
Port	Statistics of a physical port of a datapath
Queue	Statistics regarding a queue of a port
Vendor	Vendor-defined

Table 3.12: Structure of the body of the description statistics message reply.

Field	Length (bits)	Description
MFR DESC	**256**	Manufacturer description
HW DESC	**256**	Hardware description
SW DESC	**256**	Software description
Serial Num	**32**	Serial number of the datapath
DP DESC	**256**	General description of the datapath

Statistic Messages

One of the most important functions of the OpenFlow protocol is to get statistics from the datapath. The SDN controller sends a statistic request and the datapath answer with a statistic reply. Both messages indicate the type of statistics that are requested or provided and, then, the values of the statistics. The type of the statistics is defined by a 16 bits field and its possible values are described in Table. 3.11. Beside the type field, there is also a flags field that has no use in the 1.0 version of the protocol.

Statistics Description

The description type of a statistics message is used to communicate information about the datapath manufacturer, hardware and software revision, the serial number and a general description. Table. 3.12 summarizes and describes these fields, which are included in the reply messages. The request has no body, it is only composed by the type and the flags fields.

Table 3.13: Structure of the body of the flow statistics message reply.

Field	Length (bits)	Description
Length	**16**	Length of this entry
Table ID	**8**	The ID of the table the flow came from
Pad	**8**	Pad to align
Match	**320**	Fields to match and their values
Duration Sec	**32**	The time the flow has been active in seconds
Duration Nsec	**32**	Amount of nanoseconds that the flow has been active beyond the Duration sec time
Priority	**16**	Priority of the flow
Iddle Timeout	**16**	Number of seconds idle to remove the flow entry
Hard Timeout	**16**	Number of seconds to remove the flow entry
Pad2	**48**	Pad to align to 64 bits
Cookie	**64**	Controller-issued identifier
Packet Count	**64**	Number of packets in the flow
Byte Count	**64**	Number of bytes in the flow
Actions	**It depends on the action**	Actions to be performed with this flow after a match

Flow Statistics

When the SDN controller wants to gather statistics from a concrete flow, it must send in the request the fields that defines the flow (Table. 3.1) and the table ID. In the reply, the datapath answer with the fields shown in Table. 3.13. As regards statistics, the important fields are the number of packets in the flow, the bytes of the flow and the time the flow has been active. This allows calculating the bandwidth a concrete flow consumes easily.

Table 3.14: Structure of the body of the flow statistics message reply.

Field	Length (bits)	Description
Packet Count	**64**	Number of packets in the flow sett
Byte Count	**64**	Number of bytes in the flow set
Flow Count	**32**	Number of flows that matched
Pad	**32**	Align to 64 bits

Table 3.15: Structure of the body of the table statistics message reply.

Field	Length (bits)	Description
Table ID	**8**	The ID of the table
Pad	**24**	Pad to align to 32 bits
Name	**32**	Name of the table
Wildcards	**32**	Wildcards to math flows that are supported by the table
Max Entries	**32**	Max flow entries supported by the table
Active Count	**32**	Number of active flow entries
Lookup Count	**32**	Number of packets looked up in this table
Matched Count	**32**	Number of packets that matched with a flow in this table

Aggregate Statistics

The request of aggregate statistics is almost the same of the flow statistics request. The only differences are the type field of the statistics request and the structure of the reply, which is shown in Table. 3.14. However, now the match can return several flows, not only one. The reply indicates not only the packets and bytes of the set of flows but also the quantity of flows that match. This is quite useful to gather the statistics of a certain set of flows, like the multimedia flows of a network.

Table Statistics

The SDN controller can also ask for the statistics of all the flow tables of a datapath. The request has no body, while the body of the reply consists of an array with the fields represented in Table. 3.15.

Table 3.16: Structure of the body of the port statistics message reply.

Field	Length (bits)	Description
Port number	**16**	Port number
Pad	**48**	Pad to align to 64 bits
Rx Packets	**64**	Number of received packets in the port
Tx Packets	**64**	Number of transmitted packets through the port
Rx Bytes	**64**	Number of bytes received in the port
Tx Bytes	**64**	Number of bytes transmitted through the port
Rx Dropped	**64**	Number of packets dropped when received
Tx Dropped	**64**	Number of packets dropped at transmission
Rx Errors	**64**	Number of received errors
Tx Errors	**64**	Number of transmit errors
Rx Frame Errors	**64**	Number of frame alignment errors
Rx Over Errors	**64**	Number of packets with RX overrun
Rx CRC Errors	**64**	Number of packets received with a wrong CRC
Collisions	**64**	Number of collisions

Port Statistics

Gathering statistics of an individual port of a switch is a useful feature that can be easily implemented thanks to the port statistic message. The request message indicates the port number of the desired switch port from which the statistics are obtained. The fields of the reply are shown in Table. 3.16.. As regards statistics, the most interesting fields are the related to errors and packets or bytes dropped. With those statistics, the loss rate can be calculated, which is an important QoS measurement.

Table 3.17: Structure of the body of the queue statistics message reply.

Field	Length (bits)	Description
Port number	**16**	Port number
Pad	**16**	Pad to align to 32 bits
Queue ID	**32**	Identifier of the queue
Tx Bytes	**64**	Number of transmitted bytes
Tx Packets	**64**	Number of transmitted packets
Tx Errors	**64**	Number of packets that are dropped due to overrun

Table 3.18: Structure of the body of the packet-in message.

Field	Length (bits)	Description
Buffer ID	**32**	ID assigned by datapath
Total Length	**16**	Total length of the packet
In Port	**16**	Port where the packet was received
Reason	**8**	Reason why the packet was sent to the controller
Pad	**8**	Pad to align
Data	**It depends on the packet**	Packet

Queue Statistics

The OpenFlow protocol allows gathering statistics from a concrete queue of a port. The body of the reply is the same than the one used in the port statistics request, but it also contains the queue identifier. The body of the reply is shown in Table. 3.17.

Packet-In Message

When datapath receive a packet and they send it to the controller, they use the Packet-In message. This usually happens when a packet does not belong to any flow entry. However, it can be the result of an action set in the flow entry. The fields of the packet are shown in Table. 3.18.

Table 3.19: Structure of the body of the packet-out message.

Field	Length (bits)	Description
Buffer ID	**32**	ID assigned by datapath
In Port	**16**	Packet's input port
Actions Length	**16**	Length of the actions field
Actions	**8**	Actions to perform
Data	**It depends on the packet**	Packet

Table 3.20: Structure of the error message.

Field	Length (bits)	Description
Type	**32**	Type of error message
Code	**16**	Code of the error, depending on the type of error
Data	**It may vary depending on the packet**	Code of the error, depending on the type of error

Packet-Out Message

The packet-out message is the one used by the SDN controller when it wishes to send a packet out through the datapath. It is used usually as a response of a packet-In message too. The structure of the message is shown in Table. 3.19. The actions field is used to set an action to perform to that packet.

Error Message

When an error happens, the datapath needs to notify it to the controller. They use the error message to detail the error. The structure of the error message is detailed in Table. 3.20. There are two important fields in the error message, the type of error and the code, which gives more detail of the error source.

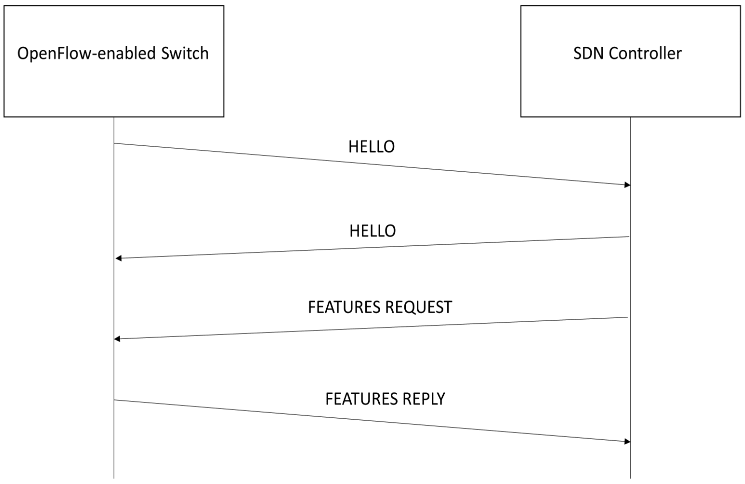


Figure 3.1: Handshake communication process.

Handshake Communication Process

In OpenFlow, the communication between a switch and an SDN controller must begin with the handshake. This process is detailed in this subsection.

The handshake communication is depicted in Fig. 3.1. In this figure, the messages exchanged between the OpenFlow-enabled switch and the SDN controller are shown. When an SDN controller is running, it listens connections to the 6533 TCP port. The switch sends to that port, and the IP address of the controller (set by configuration) a hello Openflow message. Both sides indicate the minimum version of the OpenFlow protocol they can manage. The version used will be the lower of the two values.

The hello message is not the only message exchanged in the handshake communication process. The SDN controller must ask for the features of the switch. This is done with the features request OpenFlow message. Once the switch replies with the features reply message, the SDN controller can add the switch to its list of devices and start working with it.

3.1.3 Virtualization in SDN

One of the advantages of SDN is that allows abstracting the hardware and network topology underlying. This make easier to manage the network and to create network applications and standards. Although is out of the scope of this work, network function virtualization (NFV) is a paradigm whose aim is to provide virtualization network elements. Therefore, the hardware turns into software applications, dunning in virtualized environments. NFV is studied to be applied in technologies like mobile networks.

As regards SDN, there are some emulators and network software that is usually used either in the network or in controller frameworks. Some of this software is explained in this section.

Open vSwitch

Open vSwitch, or OpenvSwitch, is a software which virtualize a switch [152]. The switches emulated are compatible not only with several traditional network protocols such as 802.1Q, STP, IPv6, GRE and so on, but also with OpenFlow.

The main components of Open vSwitch are ovs-vswitchd, which is the daemon that implements the switch, ovs-dpctl, which configures the switch kernel, ovsdb-server, which is a database server that is queried to obtain its configuration, and ovs-appctl, a utility to send commands to the Open vSwitch daemons that are running.

Moreover, Open vSwitch also provides some additional tools. The most important one is ovs-ofctl. It allows the user to control OpenFlow switches and controllers. It can be used to test the performance of a virtualized SDN. Moreover, the ovs-testcontroller is a tool composed by a simple OpenFlow controller that can be used for testing.

Open vSwitch provides an introduction of virtualization in networking [153]. However, along with SDN, it improves the capabilities of virtualization in data centers [154].

Mininet

Mininet [155] is an open-source SDN emulator. Mininet creates a realistic virtual network [156]. It runs switches and applications on a single machine. That is, a collection of end-hosts, switches, routers, and links on a single Linux kernel. The host works like a real machine and they send packets through their virtual interface. The network topology and configuration are defined by scripts in Python. Moreover, links can be configured to set a certain bandwidth, delay and loss rate. Therefore, Mininet becomes a useful tool for developers and researchers to test systems or novel applications in SDN environments.

Mininet does not provide an SDN controller. It uses POX as a default controller but if the researchers or the developers want to provide custom functionalities, they have to add an adequate controller. For instance, if a specific switching or routing is desired to be tested or to be applied to the virtualized SDN, that module must be added to a controller. Mininet allows the communication with a custom external SDN controller.

Mininet runs in Linux systems. It uses the default Linux bridge or Open vSwitch to emulate switches. Moreover, network hosts are emulated as a Linux machine, being part of the emulation host Linux system. Consequently, network hosts share the file system and process identifier (PID) space. That means the running daemons that require configuration in /etc are shared among the hosts. This is a limitation that should be solved by the user connecting several Mininet instances.

There is also a fork of Mininet that includes Wi-Fi Stations and Access Points. Its name is Mininet-Wi-Fi [157] and it works exactly as the original Mininet. It allows researchers or developers to test SDN with wireless host. Therefore, the tests can be applied to more kind of networks.

Quagga

Quagga [158] is a routing suite for Linux systems. It allows implementing several routing protocols such as OSPF or RIP to a virtualized network node [159].

The architecture of Quagga is composed by a main core daemon called Zebra [160], which abstracts the UNIX layer behind and provides an API. This API is used by the clients that implements the different routing protocols. These routing protocols run as daemons. They are configured either using a CLI or writing a configuration file.

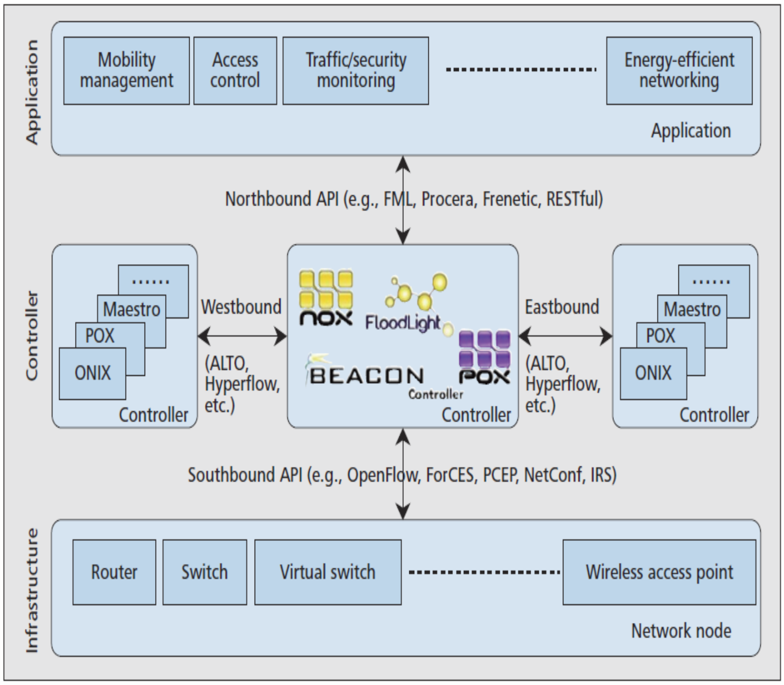


Figure 3.2: Network architecture and interfaces of an SDN.

3.2 Architecture

The architecture of the SDN controller is explained in this subsection.

Fig. 3.2 shows the details of the general architecture of an SDN. This figure depicts the different interfaces and the interaction between the SDN Controller and the other elements of the network. In the figure, can be seen two API called "Westbound" and "Eastbound" [161]. These are used to interconnect SDN controller of different domains. The main reason is to provide SDN to WAN. Nonetheless, these APIs are outbound of the scope of this work.

The general architecture diagram is shown in Fig. 3.3. Since the goal of the controller is testing, in this chapter, the Northbound API to allow external application to access to the core services is not addressed. The user can access to a GUI, which is shown in Section 3.7. However, in chapter 4, where the routing protocol is described, the possibility of using the Northbound API to place the AI services and other applications is not denied.

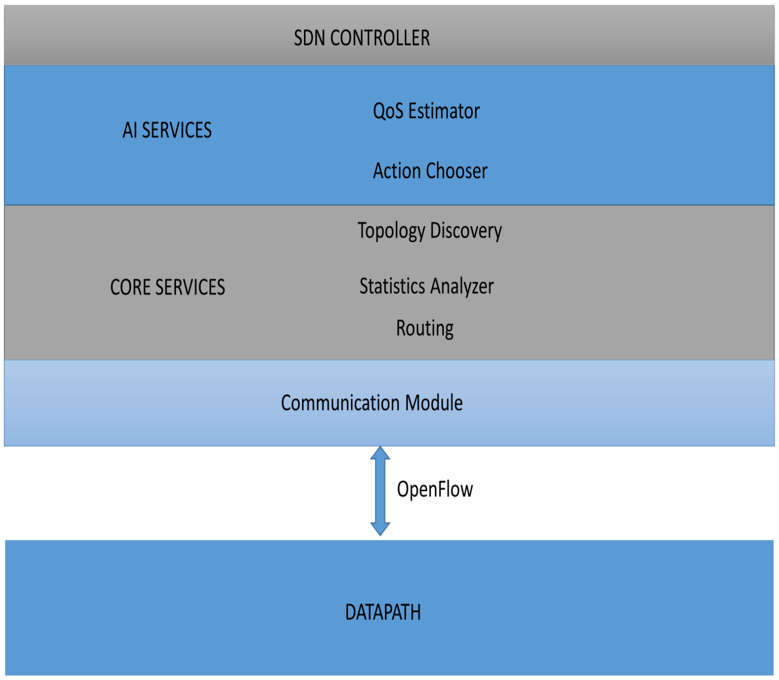


Figure 3.3: Custom SDN Controller Architecture.

There are three different modules in the architecture: the communication module, the core services and the AI services. The communication with the datapath is implemented with the OpenFlow standard. The communication module is the module which implements this communication based on the OpenFlow protocol. The core services present a topology discovery service, which creates a virtual map of the network topology. The user can also load a defined topology from a text file to make quicker the tests.

Another important core service is the statistics analyzer. It uses the communication module to register statistics from the different nodes, flow tables, flow or even ports. These statistics will be important for the AI services. The last core service is the routing module. This last service implements the routing protocol, explained in chapter 5.

3.3 Use Cases

From this subsection on, the design of the controller is presented. This design has been done following the Unified Modeling Language (UML) [162]. The functionality of the SDN controller is focused on providing a test environment. Therefore, the user cases all test-oriented.

Fig. 3.4 shows the actors and the user cases. There are four different actors. Firstly, we can find the network nodes. They can add themselves to the network, making the connection to the SDN controller, test the connectivity with the controller and ask action to perform when a packet arrives and it does not match any flow entry. The second actor is time, which triggers a stat request from the SDN controller to the network nodes. The routing module, the third actor, can perform changes, which can be network changes or behavior changes. If the change is a network change, the routing module needs to send flow mod messages to perform the actions. This can be also done by a user, who can send a single flow mod for testing purposes or can load a set of changes to be performed. The user can get the stats too or load a topology when a closed test is being performed.

Table. 3.21 shows the details of the user case "Add Node". This is the case when the network node asks the system to be added. That means, the node starts with the OpenFlow handshake and the system must add it to its list of nodes.

Table. 3.22 details the user case called "Test Connectivity". This user case is performed by a network node each time it wants to check if the SDN controller is alive. For that purpose, it uses the OpenFlow messages "Echo Request" and the SDN controller uses "Echo Reply".

Table. 3.23 details the "Ask Action" user case. The network nodes can ask the SDN controller which actions they should perform with a packet which does not match any flow entry.

Table. 3.24 describes the user case "Stats Request". This is a user case that is activated when a certain amount of time has passed. The SDN controller uses then the OpenFlow protocol to gather the statistics from the network nodes. Then, the controller can update its data.

Table. 3.25 shows the details of the user case "Load Topology". This user case allows the user to load a topology. This is oriented to perform closed tests, which have a predefined topology and the user knows the features and

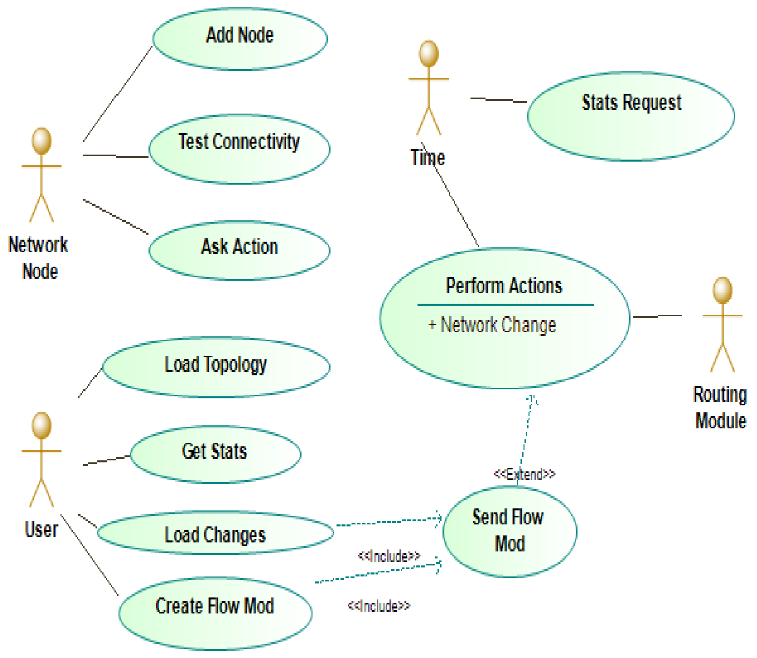


Figure 3.4: User case diagram

Table 3.21: Details of Add node user case.

User Case	Add Node	
Preconditions	None	
Postconditions	A new switch is now added to the system.	
Description	The node starts the communication with the controller and it is added to its control network.	
Event Flow	**Actors' actions**	**Response of the system**
	The node sends a Hello Message	
		The system reads the message and response with a features request.
	The node sends a features reply	
		The system adds the switch and send the default configuration.

Table 3.22: Details of Test Connectivity node user case.

User Case	Test Connectivity	
Preconditions	None	
Postconditions	None	
Description	The node sends an Echo Request to the node. If the node does not do that, the controller removes the node from the network.	
Event Flow	**Actors' actions**	**Response of the system**
	The node sends an Echo Request message.	
		The system replies with an Echo Reply message.

Table 3.23: Details of Ask Action user case.

User Case	**Ask Action**	
Preconditions	The network node has received a packet which does not match any flow entry.	
Postconditions	A new flow entry is added to the flow table of the node.	
Description	The network node asks for an action to do with a specific packet. The system responses with the action and add a new flow entry.	
Event Flow	**Actors' actions**	**Response of the system**
	The node sends a Packet In message	
		The system decides the action to be performed and sends a Packet Out and a Flow Mod message.
	The node performs the action and modifies the flow table.	

Table 3.24: Details of Stats Request user case.

User Case	**Stats Request**	
Preconditions	None	
Postconditions	The stats are updated.	
Description	After an established period of time, the system updates the statistics.	
	Actors' actions	**Response of the system**
Event Flow	Triggers the gather stats event.	
		The system asks the nodes for the statistics using OpenFlow. When the system gathers the new data, it uses them to update the statistics.

conditions of each network node. The user can provide a wrong-formatted file, which ends up in the system returning an error message.

Table. 3.26 shows the details of the user case "Get Stats". This action is performed by the user to export the current stats gathered by the SDN controller into text files. The stats are classified into Flow, Port, Node or Table stats. Therefore, a file is created for each type of stats. The user can use these files to check the network performance and extract graphs.

Table. 3.27 describes the user case "Load Changes". The SDN controller can perform a series of changes provided by the user. This is designed to quickly prepare the flow tables of the network nodes, both in a test scenario and in a working one. Since the user provide the changes in a file text, it may be wrong formatted. If this happens, the system performs no action and returns an error message.

Table. 3.28 shows the user case "Create Flow Mod". The user can create a flow mod message through the user interface of the SDN controller. The user introduces the value of each field to send the flow mod. This is a slow way to create changes in the flow table of the network nodes. However, it is quite

Table 3.25: Details of Load Topology user case.

User Case	**Load Topology**	
Preconditions	None	
Postconditions	The topology the user provided is set for the current network nodes.	
Description	The user can load a file with the topology for the network.	
Event Flow	**Actors' actions**	**Response of the system**
	The user selects the file he wants to load.	
		The system reads the file and load the topology.
Alternative Event Flow	**Actors' actions**	**Response of the system**
	The user selects the file he wants to load.	
		The system detects errors in the file format. Therefore, the system does not load any change and show a message to the user.

Table 3.26: Details of Get Stats user case.

User Case	**Get Stats**	
Preconditions	None	
Postconditions	Files are written with the stats.	
Description	The stats are exported into several files.	
Event Flow	**Actors' actions**	**Response of the system**
	The user requests the current updated stats.	
		The system gets the current data and writes them into files, attending to at which element the data refers to.

useful to test the capabilities of the system. If the user introduces wrong values, the system does not send the message and returns an error.

Table. 3.29 describes the user case "Send Flow Mod". It describes the process where the SDN controller sends the OpenFlow "Flow Mod" message to a specific network node. This message can be provided either by the routing module or by the user.

The last user case, "Perform Action", is described in Table. 3.30. This user case is performed by the routing module. When the routing algorithm decides what action should be taken, this is notified to the SDN controller. After that, the system performs the action. If it is needed, an Openflow Flow Mod is sent to modify the flow tables on the nodes. The working of the routing protocol is explained in the chapter 4.

3.4 Component and Class Diagrams

Once the user cases have been designed, the components and the classes that implement the functionality defined in these user cases are created. In this section, the diagrams containing their design are depicted and explained.

Table 3.27: Details of Load Changes user case.

User Case	**Load Changes**	
Preconditions	None	
Postconditions	The flow tables of the nodes contain the flows provided by the user.	
Description	The user provides a text file with flows that the SDN controller add to the tables of the nodes.	
	Actors' actions	**Response of the system**
Event Flow	The user provides a file with changes in the network nodes' flow tables.	
		The system reads each line of the file and perform the actions written there.
	Actors' actions	**Response of the system**
Alternative Event Flow	The user provides a file with changes in the network nodes' flow tables, but with a wrong format.	
		The system does not perform the actions and returns an error message.

Table 3.28: Details of Create Flow Mod user case.

User Case	**Create Flow Mod**	
Preconditions	None	
Postconditions	A "Flow Mod" message is sent to a network node.	
Description	The user introduces the values of a "Flow Mod" message and the SDN controller send the message to the selected network node.	
Event Flow	**Actors' actions**	**Response of the system**
	The user selects a network node.	
		The system shows a user interface where the user can introduce the values of a new "Flow Mod" message.
	The user introduces the values of the messages and click the "Send" button.	
		The system sends the message to the network node.
Alternative Event Flow	**Actors' actions**	**Response of the system**
	The user introduces wrong values for the "Flow Mod" message.	
		The system does not send the message and returns an error message.

Table 3.29: Details of Send Flow Mod user case.

User Case	Send Flow Mod	
Preconditions	None	
Postconditions	A Flow Mod is sent to a network node.	
Description	The SDN controller sends an Openflow Flow Mode message that the routing module or the user has provided.	
	Actors' actions	**Response of the system**
Event Flow	The user or the routing module provides the system with the flow mode and with the destination network node.	
		The SDN controller search the destination network node. The system sends the Openflow Flow Mod to the selected node.

Table 3.30: Details of Perform Action user case.

User Case	**Perform Action**	
Preconditions	None	
Postconditions	None	
Description	The system receives an action from the routing module and performs it.	
	Actors' actions	**Response of the system**
Event Flow	The routing module indicates the recommended action to the controller.	
		The system reads the action and executes it. If it is required, the system may send Openflow Flow Mod messages.

On the one hand, Fig. 3.5 depicts the component diagram. In this diagram, the different software blocks of the SDN controller are shown. Furthermore, the classes that compose the components are listed and the relations between the components are drawn. As can see in the picture, the SDN controller developed is composed by the following modules:

- Core: it is the main component. It implements the SDN controller server, the topology management, contains structures to store and manage data of switches and events, implements the user interfaces described in section 3.6, manage the statistics and provide the interface for the communication and for the routing.
- Openflow: this is the component that implements the OpenFlow protocol. All the types of messages and structures needed from the OpenFlow 1.0 standard are located in this package. When the SDN controller needs to send a message to communicate with the nodes, it creates messages by using the classes of this component.
- JSON: in order to implement some functionalities, like loading a topology or a list of events ("load changes" user case), the JSON text format is used

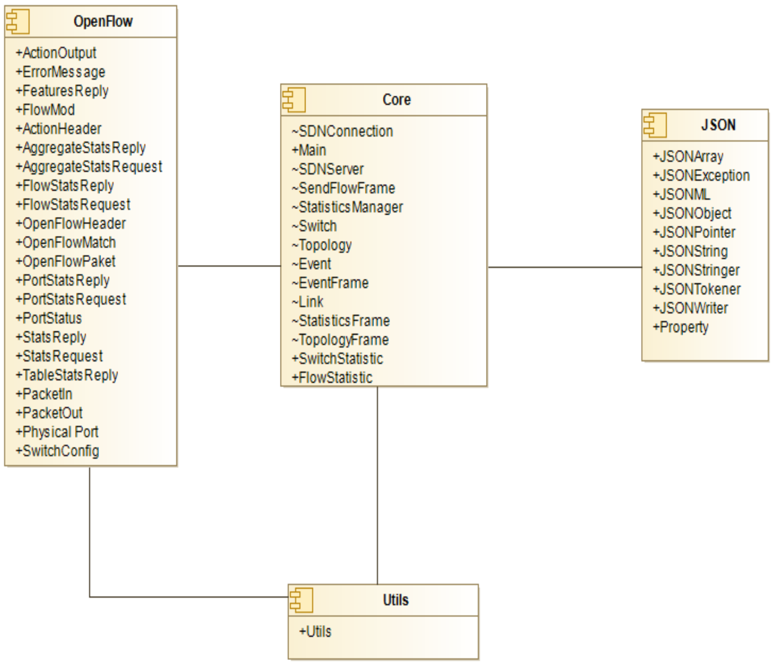


Figure 3.5: Component Diagram.

(see[163] and [164]). JSON offers an easy way to parse the documents into Java objects. This parsing procedure has been done with the org.json library [165].

- Utils: contain utilities, like constants and configuration parameters of the SDN controller.

This modular design allows replacing any of the components or adding more easily. For instance, in order to add compatibility with OpenFlow 1.3, the component OpenFlow would be replaced with a similar module with the classes needed to implement the 1.3 version of the standard or would be completed with those classes. It would be the same if in the future the parsing method would be changed. Only the JSON component would be replaced.

On the other hand, Fig. 3.6 shows the class diagram of the core component. In this diagram, the classes, some of their most important fields and methods and their relationships with other classes are depicted.

Before starting depicting the classes that compose the diagram, some points must be taking into account:

- The class diagram refers only to the core component. Utils is a single-class component and JSON is an external library whose design is not related to this work. The OpenFlow component contains the classes for the structures and packets described in chapter 1. They also contain methods to write these messages in a socket and to read them from a socket.
- The classes only show the most important attributes and methods in order to save space and guarantee that the picture can be read. All the get and set methods are not depicted, and the attributes used to the internal working do not appear. The constructors are also omitted. Some names of the methods are shorter than in the final implementation in order to save space.
- Some relationships with other classes do not appear in the diagram to save space. This is usually related to the packets the classes used, the use of the Utils class or the use of the JSON library to parse the files. However, the "StatisticsManager" class has some relationship that are indicated as attributes. These omissions do not affect the sequence diagrams detailed in the next section.
- Although the OpenFlow-component class diagram is not represented, all the packets inherits from "OpenFlowPacket" class. Therefore, this class can be referenced to indicate that any kind of packet can be used as attribute or argument. "StatsReply" and "StatsRequest" classes are also super classes for statistics.

As regards the classes, the most important ones are:

- Main: it is the class that contains the relations with all the other classes, the first point in the program and it also implements the first frame, the first user interface.
- SDNServer: implements the server, which interacts with the nodes and with the other components and applications (implements the interfaces).
- SDNConnection: each connection from the server to a node or a set of nodes. It is used to perform the communication. It contains the sockets to send the messages.

- Frame classes: they are the classes that implements the user interfaces.
- Topology: the class that stores the topology. Contains a list of nodes and links between those nodes, with information about the performance of those links and their status.
- Switch: the information about the nodes proportionated during the handshake procedure it is stored there. Furthermore, an id is assigned.
- Event: this class is created to implement changes in the network. The event contains a time when it will be activated and a list of packets and nodes (destinations) to perform those changes.
- StatisticsManager: the statistics are requested and processed by the "StatistcsManager" class. Each certain amount of time, this class send statistics request to the nodes. The nodes reply with the statistics at that time. Then, the "StatisticsManager" object stores these data and process them to extract the statistics used to build the graphs in the experiments.

3.5 Sequence Diagrams and Implementations Fragments

In this subsection, the sequence diagrams are shown and described. These diagrams show a general description of the implementation of the user cases. Each diagram corresponds to one user case. In these diagrams, not only the lifetime of the different classes that interacts in the user case appear, but also, in some of them, the communication messages are depicted to. These messages are implemented with sockets.

Fig. 3.7 shows the sequence diagram of the "Add node" user case. In this user case, a network node asks to be added to the network. In the diagram, the different messages and methods are depicted. First, the node sends a hello message to the system. This message is received by a new "SDNConnection" object that the "SDNServer" instance created when the socket was initialized. This "SDNConnection" object creates then a new "OpenFlowHeader" object to represent the reply, a new hello message. This object is sent to the node as a reply. After sending the hello message, the "SDNConnection" object sends a "Features Request" message, which is represented by an OpenFlow header too. The node responds with a "Features Reply" message, which contains the capabilities of the node. The "SDNConnection" object stores the data into a "FeaturesReply" object. After that, it creates a "Switch" object, with an id and that contains the capabilities described in the "FeaturesReply" object.

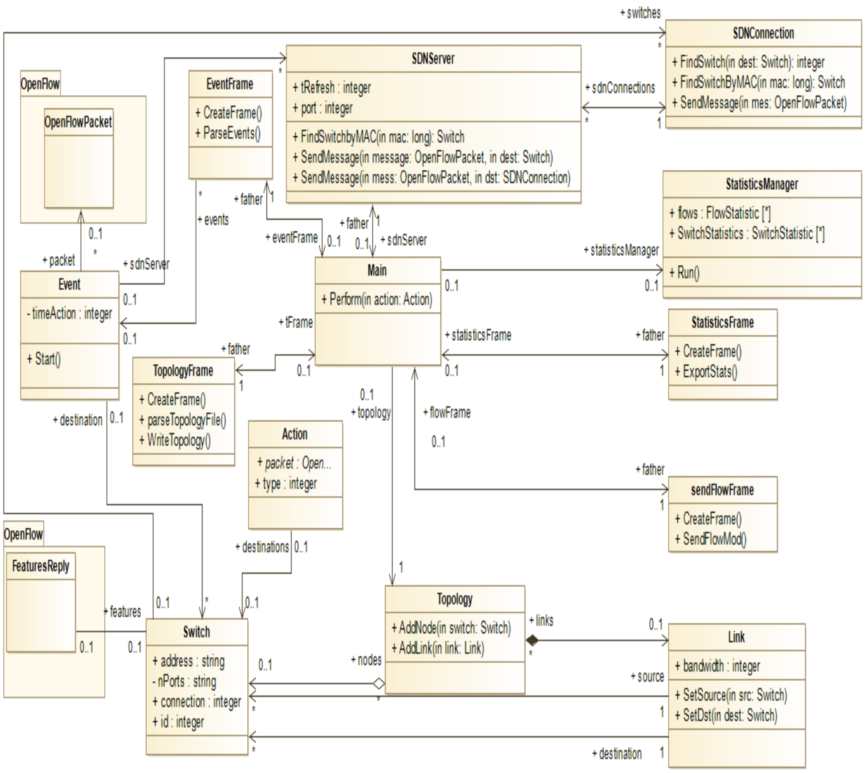


Figure 3.6: Class Diagram.

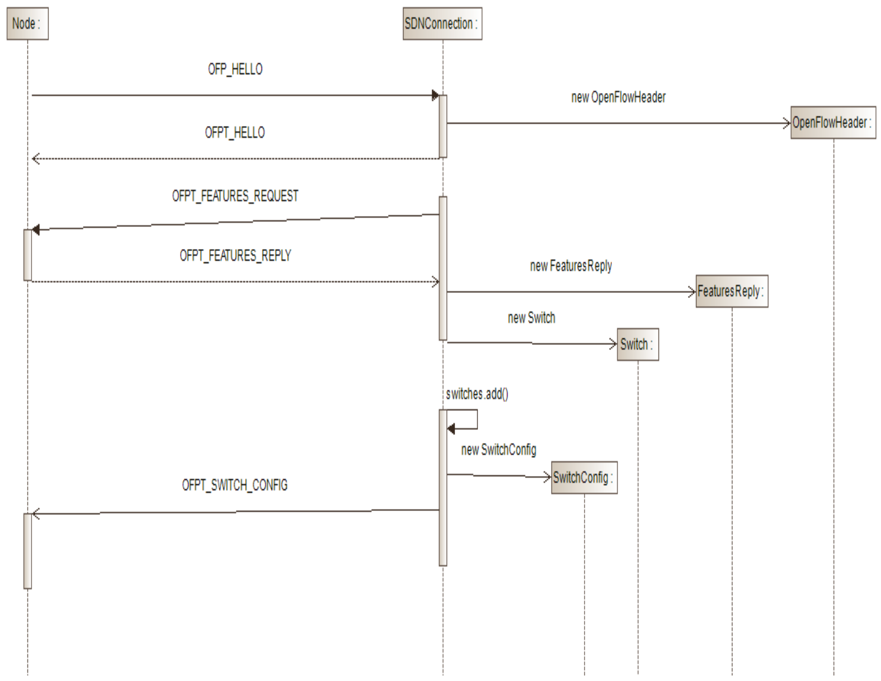


Figure 3.7: "Add node" sequence diagram.

Finally, the switch is added to the list and a “SwitchConfig” object is created to implement the “Switch Config” message, which is sent to the node to set the default configuration of the node.

Fig. 3.8 shows the sequence diagram of the “Test Connectivity” user case. The nodes can check whether the SDN controller is available or not. In order to implement this, the “Echo Request” and “Echo Reply” OpenFlow messages are used. Once again, an object of the “SDNConnection” class (now an existent one) is the one which receives the message and answer to the switch with a new “OpenFlowHeader” object, with the fields set to the values that the standard specifies for an “Echo Reply” message. This message is sent to the node.

Fig. 3.9 describes the “Ask Action” user case. This user case is presented when the switch, the network node, receives a packet and this packet does not match any flow entry. It will depend on the implementation of the routing protocol. One possible option it would be define a default action to perform. Another solution would be asking to the routing module which action should be performed. Regardless the policy of the routing protocol, a new “Action

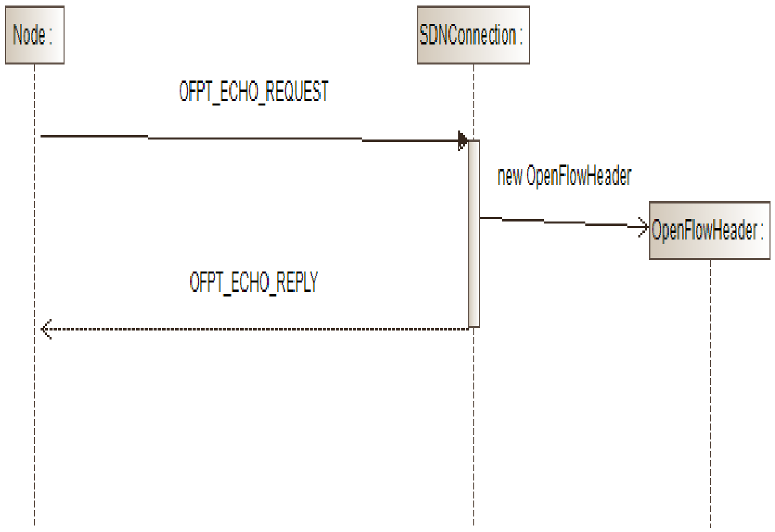


Figure 3.8: "Test connectivity" sequence diagram.

Header" object is created and inserted into a "PacketOut" object. This object is sent to the node as a "Packet Out" message. With this message, the node will know what action perform with all the packets that match the new flow.

Fig. 3.10 describes the "Load Topology" user case. The sequence diagram starts with the user invoking the "ParseTopologyFrame" method on the "TopologyFrame" class. This is done through the frame, through the user interface. When this method is invoked, the "TopologyFrame" instance starts parsing the JSON file that contains the description of the topology. An example of a topology described in JSON is shown in Fig. 3.11. The code that parses this file into Java objects is shown in Fig. 3.12. This part has been excluded from the sequence diagram for two reasons. Firstly, because it depends on the JSON library. Secondly, because including this part would make the sequence diagram hard to read.

As regards the sequence diagram, in Fig. 3.10 it is depicted than, when the "TopologyFrame" parses the JSON file, it creates "Switch" instances, one for each one described in the file. Then, this instance is added to the "Topology" object. After that, it repeats the process with the links, adding the switches described in the topology.

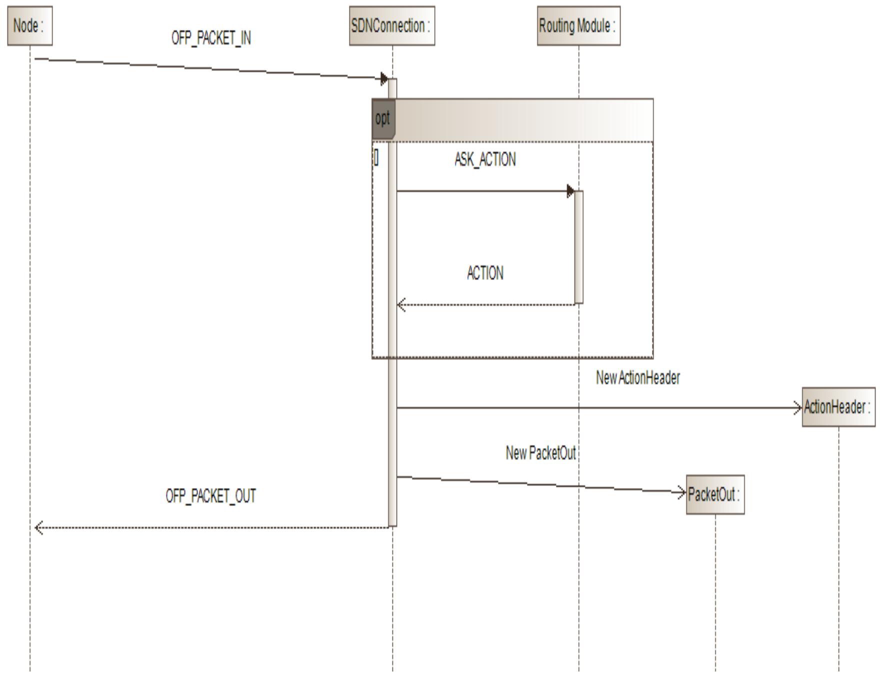


Figure 3.9: "Ask Action" sequence diagram.

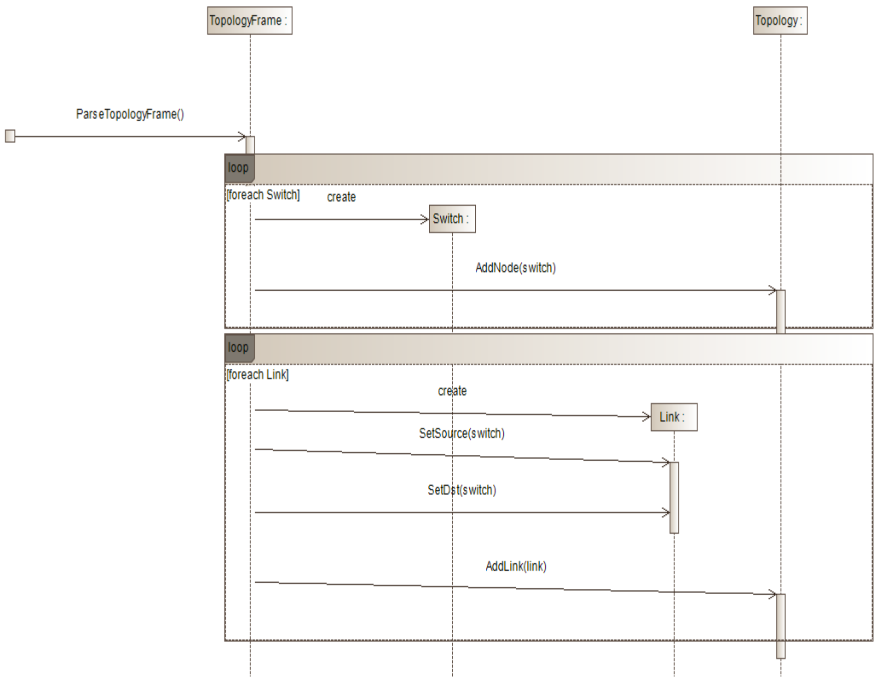
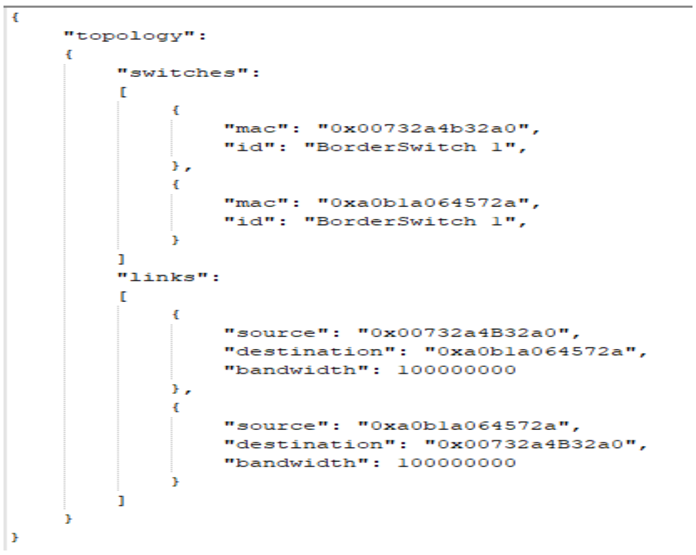
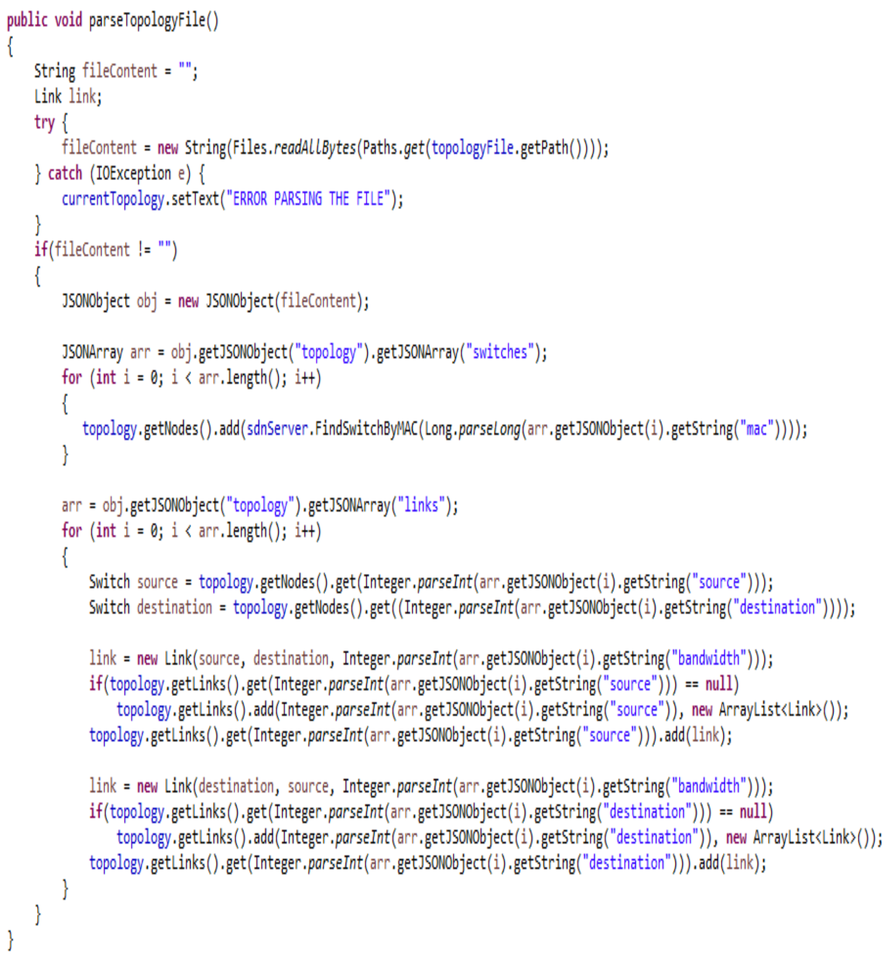


Figure 3.10: "Load Topology" sequence diagram.



```
{
    "topology":
    {
        "switches":
        [
            {
                "mac":  "0x00732a4b32a0",
                "id":  "BorderSwitch 1",
            },
            {
                "mac":  "0xa0b1a064572a",
                "id":  "BorderSwitch 1",
            }
        ]
        "links":
        [
            {
                "source":  "0x00732a4B32a0",
                "destination":  "0xa0b1a064572a",
                "bandwidth":  100000000
            },
            {
                "source":  "0xa0b1a064572a",
                "destination":  "0x00732a4B32a0",
                "bandwidth":  100000000
            }
        ]
    }
}
```

Figure 3.11: "Load Topology" JSON.



```
public void parseTopologyFile()
{
    String fileContent = "";
    Link link;
    try {
        fileContent = new String(Files.readAllBytes(Paths.get(topologyFile.getPath())));
    } catch (IOException e) {
        currentTopology.setText("ERROR PARSING THE FILE");
    }
    if(fileContent != "")
    {
        JSONObject obj = new JSONObject(fileContent);

        JSONArray arr = obj.getJSONObject("topology").getJSONArray("switches");
        for (int i = 0; i < arr.length(); i++)
        {
           topology.getNodes().add(sdnServer.FindSwitchByMAC(Long.parseLong(arr.getJSONObject(i).getString("mac"))));
        }

        arr = obj.getJSONObject("topology").getJSONArray("links");
        for (int i = 0; i < arr.length(); i++)
        {
            Switch source = topology.getNodes().get(Integer.parseInt(arr.getJSONObject(i).getString("source")));
            Switch destination = topology.getNodes().get((Integer.parseInt(arr.getJSONObject(i).getString("destination"))));

            link = new Link(source, destination, Integer.parseInt(arr.getJSONObject(i).getString("bandwidth")));
            if(topology.getLinks().get(Integer.parseInt(arr.getJSONObject(i).getString("source"))) == null)
                topology.getLinks().add(Integer.parseInt(arr.getJSONObject(i).getString("source")), new ArrayList<Link>());
            topology.getLinks().get(Integer.parseInt(arr.getJSONObject(i).getString("source"))).add(link);

            link = new Link(destination, source, Integer.parseInt(arr.getJSONObject(i).getString("bandwidth")));
            if(topology.getLinks().get(Integer.parseInt(arr.getJSONObject(i).getString("destination"))) == null)
                topology.getLinks().add(Integer.parseInt(arr.getJSONObject(i).getString("destination")), new ArrayList<Link>());
            topology.getLinks().get(Integer.parseInt(arr.getJSONObject(i).getString("destination"))).add(link);
        }
    }
}
```

Figure 3.12: "Load Topology" parsing code fragment.

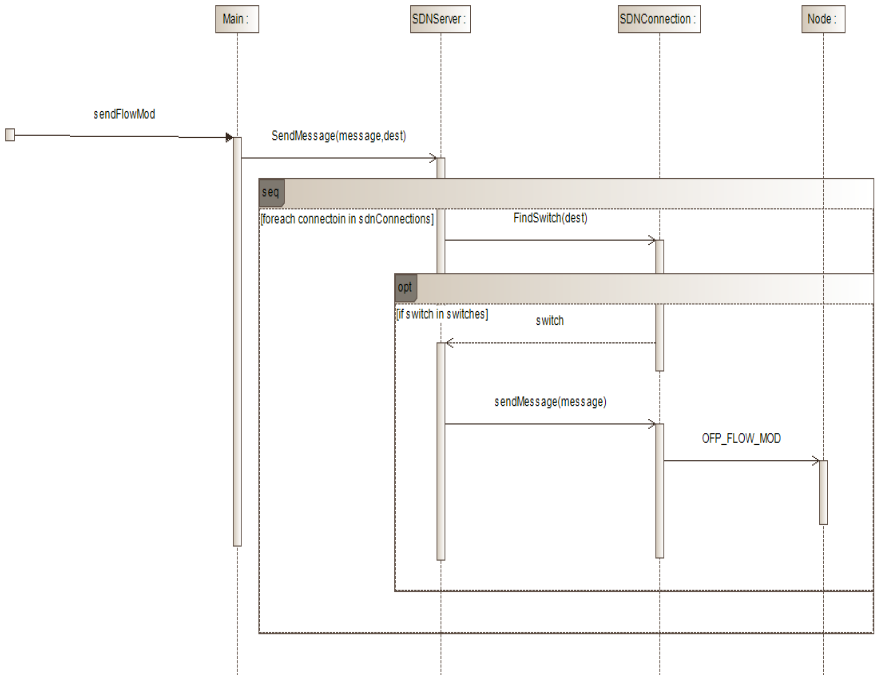
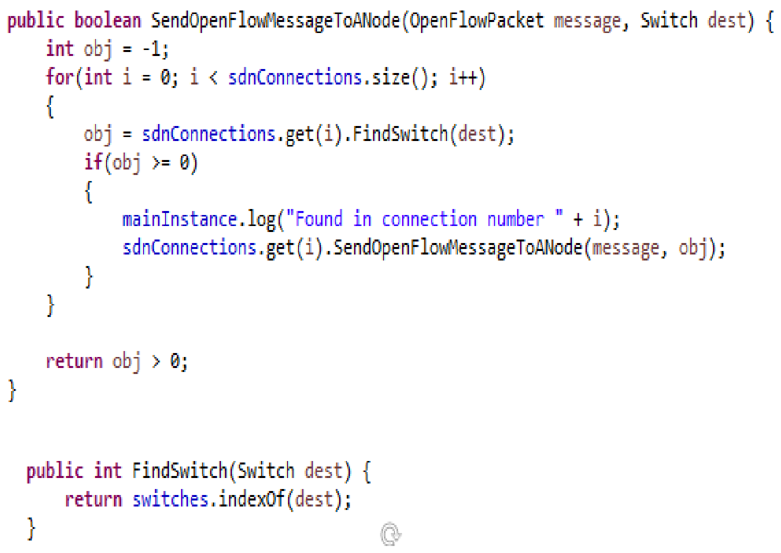


Figure 3.13: "Send Flow" sequence diagram.

Fig. 3.13 shows the sequence diagram of the "Send Flow Mod" user case. This user case is included in other user case and it represents the functionality of sending a message, in this case a "Flow Mod" OpenFlow message, to a determined node in the network. When this action is received, by invoking the "SendFlowMod" method in the "Main" class, the method "SendMessage" of the "SDNServer" is called. The arguments of the method are the message that the system want to deliver and the destination. The task of the server is searching which is the connection of the destination. In this case, it starts checking the different "SDNConnection" instances and ask them if the destination switch is one of the nodes of that connection. In that case, it writes the message in the socket and it stops searching. This stop helps to reduce the computation complexity of the algorithm. This process is shown in Fig. 3.14, where the methods "SendMessage" of the "SDNServer" class and "FindSwitch" of the "SDNConnection" class are shown. The "SendMessage" method is one of the methods whose name is shortened in the class diagram.



```
public boolean SendOpenFlowMessageToANode(OpenFlowPacket message, Switch dest) {
    int obj = -1;
    for(int i = 0; i < sdnConnections.size(); i++)
    {
        obj = sdnConnections.get(i).FindSwitch(dest);
        if(obj >= 0)
        {
            mainInstance.log("Found in connection number " + i);
            sdnConnections.get(i).SendOpenFlowMessageToANode(message, obj);
        }
    }

    return obj > 0;
}


  public int FindSwitch(Switch dest) {
      return switches.indexOf(dest);
  }
```

Figure 3.14: "Flow Mod" implementation fragment.

Fig. 3.15 shows the sequence diagram of the “Perform Action” user case. In this user case, the routing protocol notifies to the SDN controller that a certain action must be implemented. However, it depends on the routing protocol design, this action could need a “Flow Mod” message or not. It may be implemented changing some configuration parameters or modifying the behavior of some applications through the northbound API. However, if it implies network changes, these are performed sending “Flow Mod” messages, in a process described by the last sequence diagram.

Fig. 3.16 describes the diagram of the “Create Flow Mod” user case. The user accesses to a different frame, a different user interface, to insert the data of a “Flow Mod” OpenFlow message. When the user is done, invokes the “Send-FlowMod” method. This method gathers all the data from the user and creates the required structures to send the message. It creates an “OpenFlowHeader” object, an “OpenFlowMatch” that filters the flow, and the “Flow Mod” object, which represents the message. Once the data introduced by the user has been processed, the rest of the sequence matches the one explained in Fig. 3.13. Fig. 3.17 describes the “Load Events” user case. This user case is similar to “Load Topology”. The user uses a frame to load a JSON file with a list of “Flow Mod” messages. Fig. 3.18 shows an example of a JSON file used to load changes in the network. The SDN controller parses the JSON file and creates

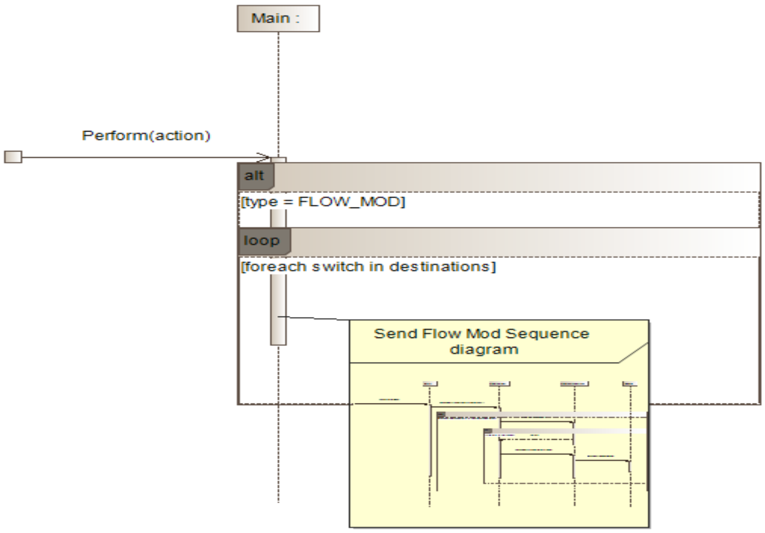

Main :
Perform(action)
alt
[type = FLOW_MOD]
loop
[foreach switch in destinations]
Send Flow Mod Sequence diagram

Figure 3.15: "Perform Action" sequence diagram.

events for each change. These events have a timer (that appear in the JSON file). Each event is a Java thread that sleep that amount of time. When the thread awakes, it sends the Flow Mod that was parsed from the JSON file. In order to do that, the procedure described in Fig. 3.11 is used.

Fig. 3.19 shows the sequence diagram of the “Request Stats” user case. This user case represents a functionality that is executed each certain amount of time. In order to do that, the “StatisticsManager” class runs in a separate thread and it awakes to send the requests to the nodes. It asks to the “SDNServer” object for the available connections and for each connection it gets the nodes. Then, the “StatisticsManager” object request stats for each node and for each port the node has. Furthermore, the “StatisticsManager” may contain a set of flows which are monitored constantly (this depends on the routing protocol). Consequently, if those flows exist, the “StatisticsManager” object will send “Flow Statistics Request” messages too. When the nodes reply with the statistics, the “StatisticsManager” stores the data.

Fig. 3.20 depicts the sequence diagram of the “Get Stats” user case. This is the user case that represents the export procedure of the statistics gathered by the “StatisticsManager” class. This class is consulted when the user, by the user interface of the “StatisticsFrame” class, activates the export of the statistics. Then, all the data that is gathered from the nodes is exporter to several files.

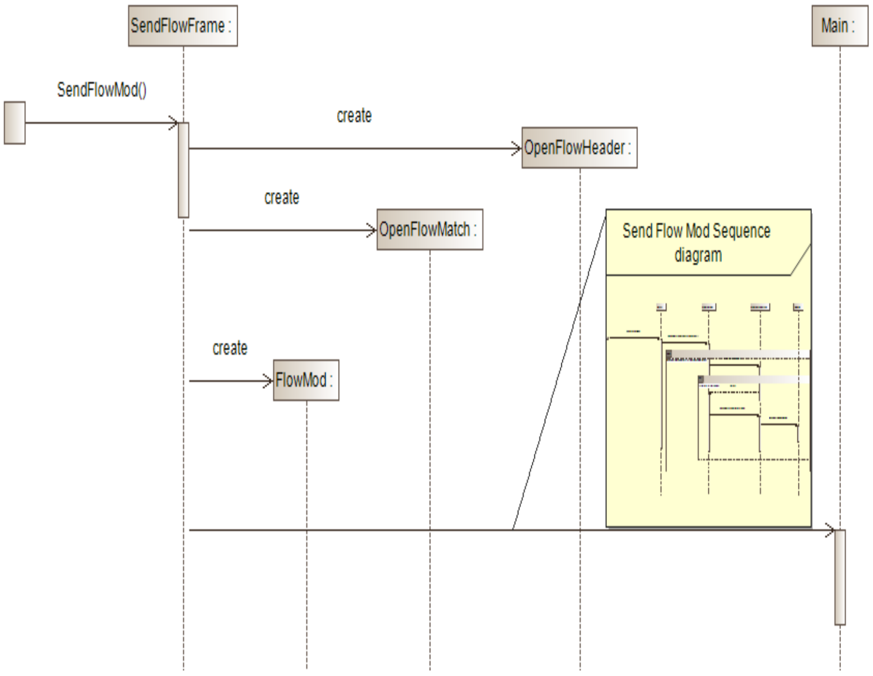


Figure 3.16: "Create Flow Mod" sequence diagram.

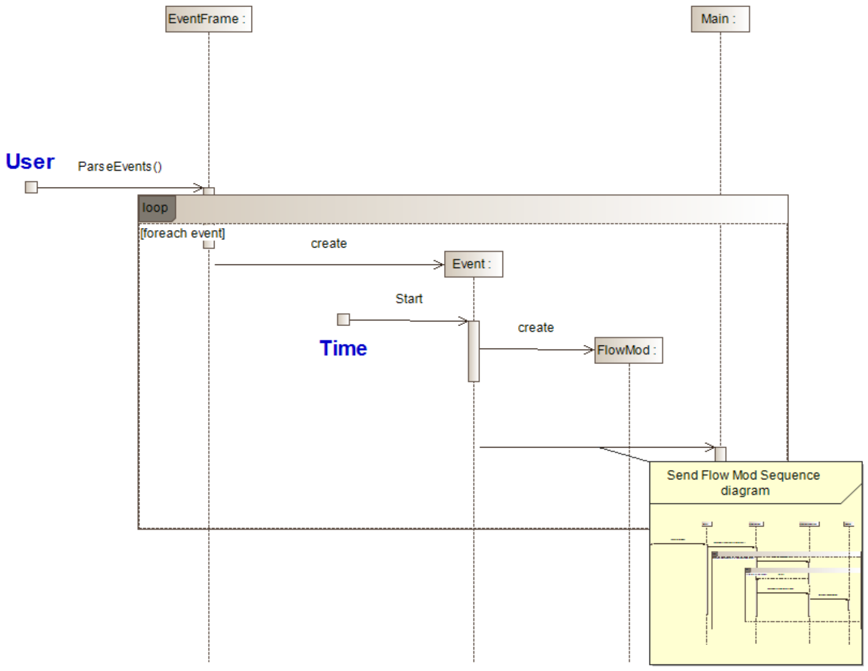
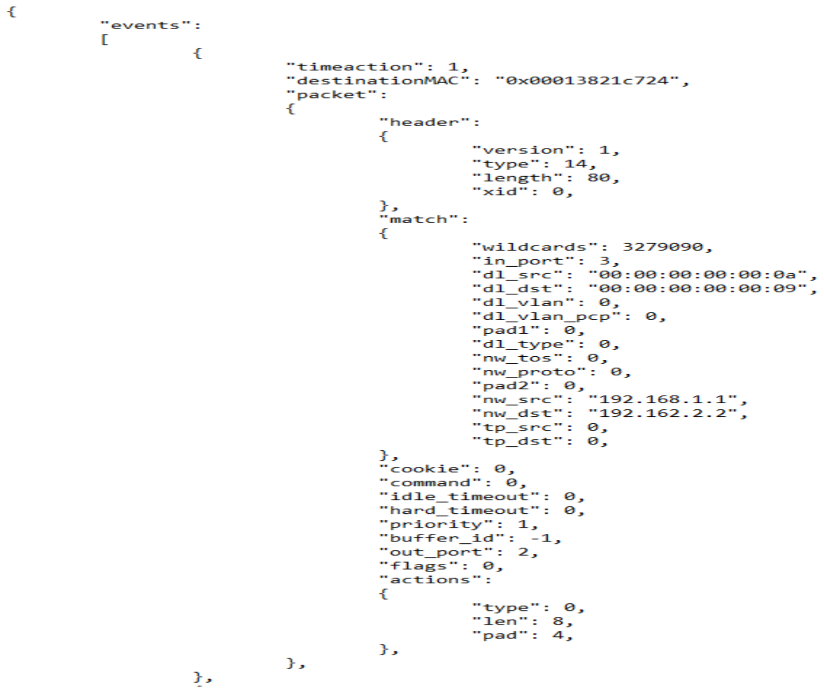


Figure 3.17: "Load Changes" sequence diagram.



```
{
        "events":
        [
                {
                        "timeaction": 1,
                        "destinationMAC": "0x00013821c724",
                        "packet":
                        {
                                "header":
                                {
                                        "version": 1,
                                        "type": 14,
                                        "length": 80,
                                        "xid": 0,
                                },
                                "match":
                                {
                                        "wildcards": 3279090,
                                        "in_port": 3,
                                        "dl_src": "00:00:00:00:00:0a",
                                        "dl_dst": "00:00:00:00:00:09",
                                        "dl_vlan": 0,
                                        "dl_vlan_pcp": 0,
                                        "pad1": 0,
                                        "dl_type": 0,
                                        "nw_tos": 0,
                                        "nw_proto": 0,
                                        "pad2": 0,
                                        "nw_src": "192.168.1.1",
                                        "nw_dst": "192.162.2.2",
                                        "tp_src": 0,
                                        "tp_dst": 0,
                                },
                                "cookie": 0,
                                "command": 0,
                                "idle_timeout": 0,
                                "hard_timeout": 0,
                                "priority": 1,
                                "buffer_id": -1,
                                "out_port": 2,
                                "flags": 0,
                                "actions":
                                {
                                        "type": 0,
                                        "len": 8,
                                        "pad": 4,
                                },
                        },
                },
```

Figure 3.18: "Load Changes" JSON.

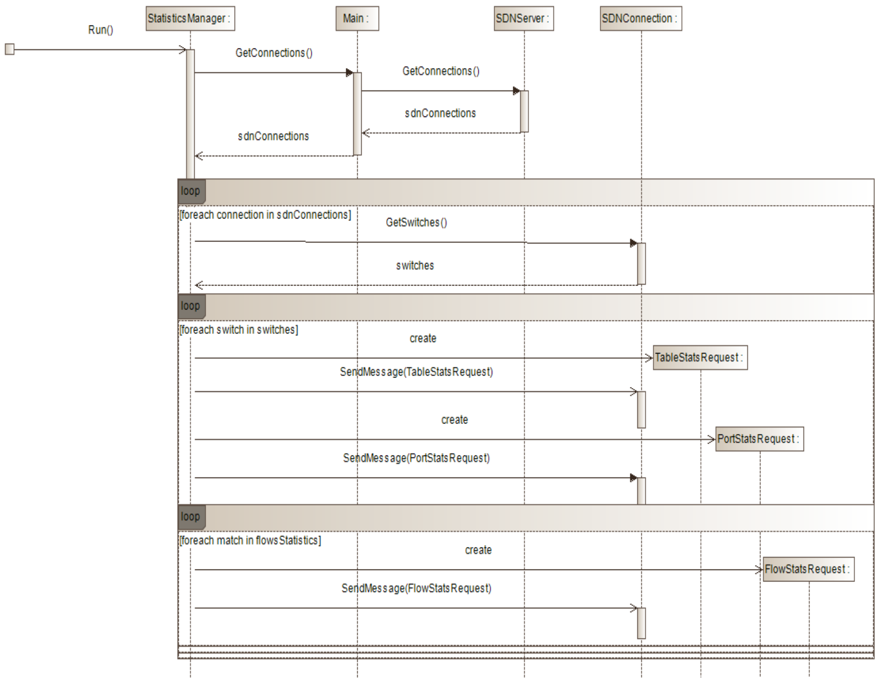


Figure 3.19: Request Stats sequence diagram.

In order to make easier the analysis of the data, the statistics for the same flow are exported to the same file, and the statistics obtained from the nodes are divided, each file contains the data of a single node.

These sequence diagrams represent the functionality in the design phase of the SDN controller. Some of this functionality needs the interaction of the user. This interaction, implemented by the frame classes, is produced through user interfaces. The design and implementation of these user interfaces is discussed in Section 3.7.

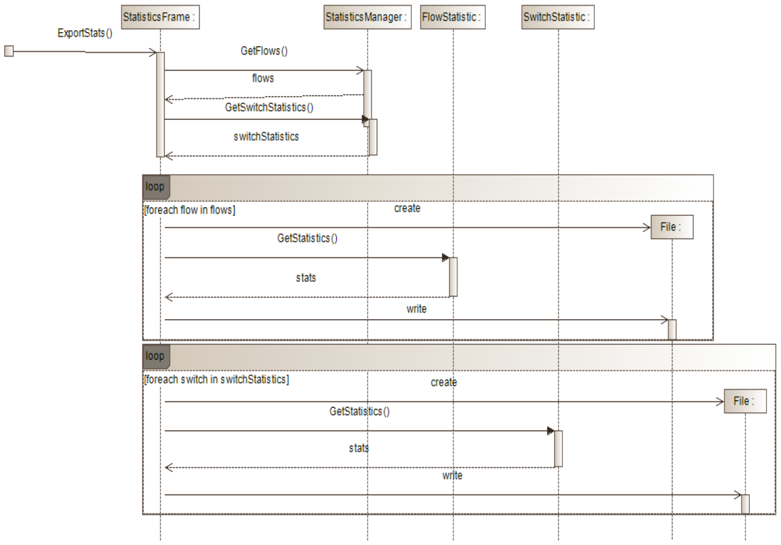


Figure 3.20: Get Stats sequence diagram.

3.6 Topology Discovery Process

In this section, the mechanism used for the SDN controller to discover the network topology is described. This is one of the requirements of every SDN controller and it plays an extremely important role in the routing proposal described in the next chapter.

In order to discover the topology, the SDN controller uses the Link Layer Discovery Protocol (LLDP) and the Address Resolution Protocol (ARP). The LLDP is used by the switches to exchange information between them. This can be used to discover the neighbors of a specific switch. Therefore, when the SDN receives a “Features Reply” message, and the handshake is over, it sends a “Flow Mod” message to force the switch to send every LLDP message to the controller. Hence, when a LLDP message is sent from a switch to the controller, the SDN controller add a link between those switches in the topology.

With this mechanism, the SDN controller knows every connection between two switches. However, in order to discover the hosts, the ARP protocol is needed. Following a procedure like the one regarding LLDP, the controller sends a “Flow Mod” message to force the switch to send the ARP requests and replies to the SDN controller. When the “Packet In” message with the ARP is received, the SDN controller add to a custom-made ARP table the mac and IP of the host, along with the port where the new host is connected.

Both mechanisms can be complemented with a “Packet Out” message sent to each switch for each protocol. This forces the switches and hosts to send the LLDP and ARP packets. This is useful when the controller is not ready to work with already-working networks or the switches are not programmed to use the LLDP by default.

With this information, the topology is automatically built. This mechanism can work together with the load topology user case. When the topology is loaded by the user, the previous topology is overwritten.

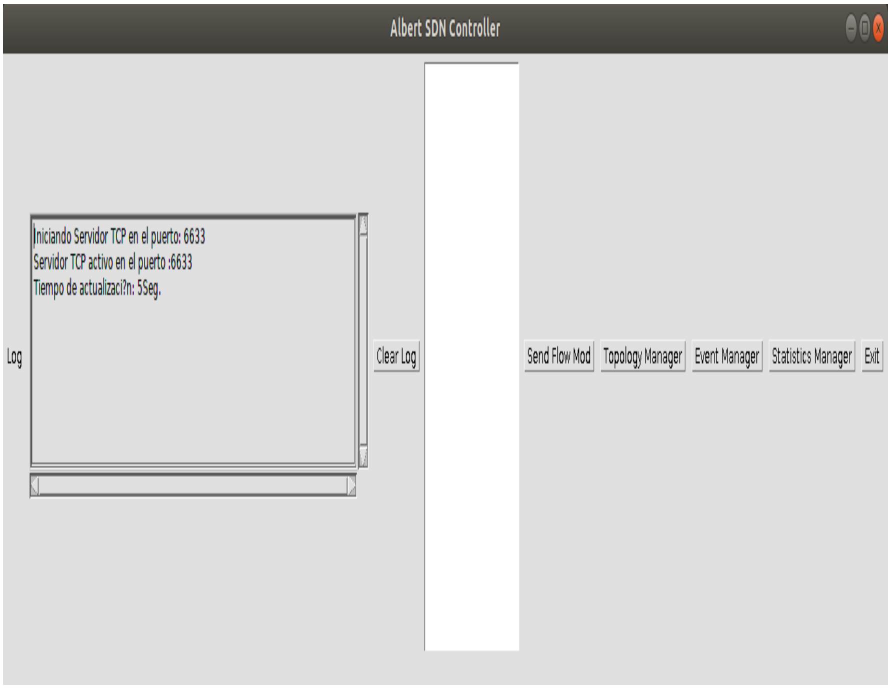
3.7 User Interfaces

The user interfaces are added to the SDN controller to make easier the operation of the user. They were designed taking into account the final goal of the SDN controller, which is to provide an easy-to-use tool to test SDN networks and solutions.

Fig. 3.21 shows the main interface of the SDN controller. It is composed by a log textbox, where the user can see different messages about server status, messages received and errors during execution. There is a button to clear the log. Next to the button, it appears a textbox with a list of nodes. This list is filled during the execution, with the nodes in the network. In Fig. 3.22, some nodes are shown. The SDN give an ID to the nodes and the MAC also appears in the textbox. Lastly, there are some buttons. The last one is to close the SDN controller.

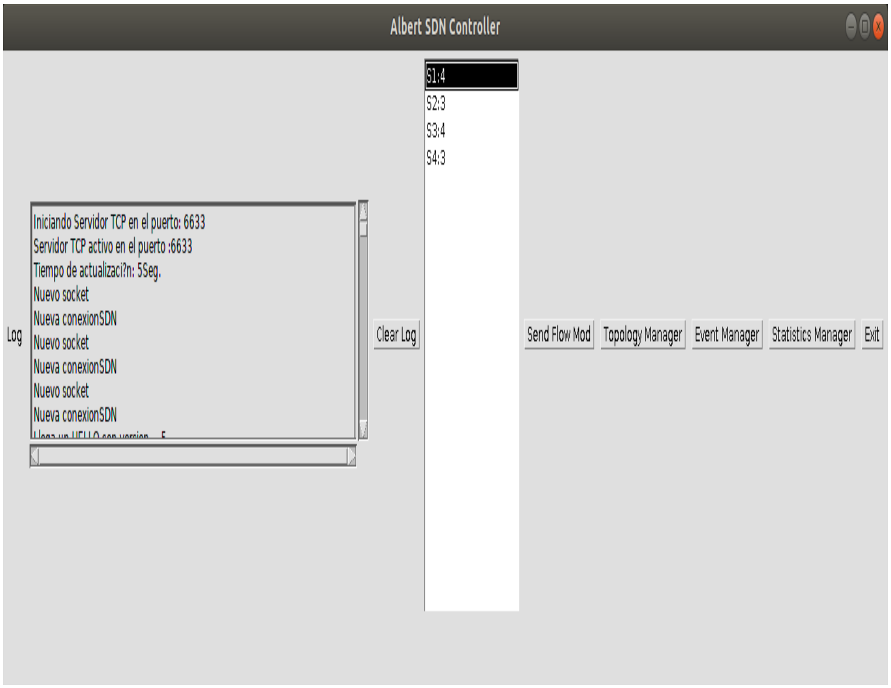
The rest of the buttons open other user interfaces. The first one is used to send an OpenFlow flow mod message to a specific node. The node is selected, like in Fig. 3.22, and then, the user clicks the “Send Flow Mod” button. The other user interfaces do not need a node to be selected.

If the user presses the “Send Flow Mod” button with a node selected, the user interface of Fig. 3.23 is shown. There, some formularies are presented to the user so that they can use them to prepare a flow mod message. Due to the huge quantity of fields that message has, only the most important ones are shown. Despite the fact that the user then can test the behavior of the nodes, this interface is mainly used for development issues.



```
Iniciando Servidor TCP en el puerto: 6633
Servidor TCP activo en el puerto :6633
Tiempo de actualizaci?n: 5Seg.
```

Figure 3.21: Main interface.



```
Iniciando Servidor TCP en el puerto: 6633
Servidor TCP activo en el puerto :6633
Tiempo de actualizaci?n: 5Seg.
Nuevo socket
Nueva conexionSDN
Nuevo socket
Nueva conexionSDN
Nuevo socket
Nueva conexionSDN
```

Figure 3.22: Main interface with a list of nodes.

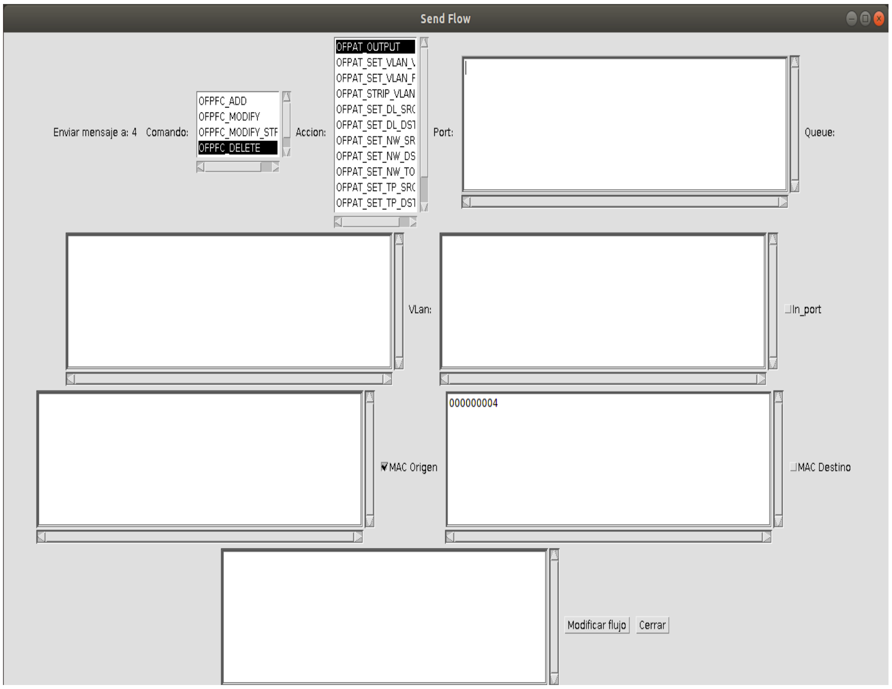


Figure 3.23: Send Flow Mod interface.

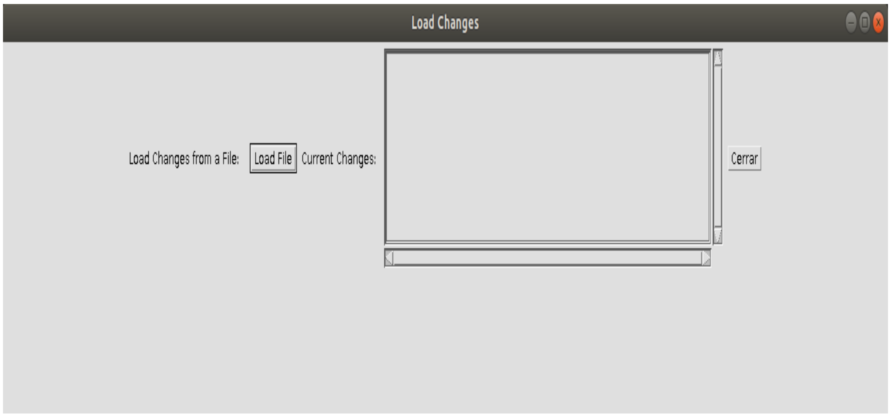


Figure 3.24: Load Changes interface.

In order to test an entire network or specific systems or algorithms, the load changes user interface is introduced. This interface, shown in Fig. 3.24, allows the user to load a JSON file to perform a set of changes in the network. The JSON file format is shown in Fig. 3.18. Each change in the network is defined by an OpenFlow packet, set to a destination in an interval of time. The user can prepare all the changes they need in advance and test it loading the changes from the JSON.

When the user clicks the “Load File” button, a new window pops up to allow the user to load the file. This window is shown in Fig. 3.25. The user selects the file they want to use. Then, the “Current Changes” textbox is filled with the changes performed by the controller. If some error with the values is found, it will be prompted in the textbox. Likewise, if the file contains format errors, a message is shown in the textbox and no changes are performed. The user, then, can correct the mistakes and load the file again.

Another user interface, shown in Fig. 3.26, is the Load Topology interface. The user can open this interface through the “Topology Manager” in the main interface. The interface provides a textbox called “Current Topology”, that shows the topology of the network, and a “Load Topology” button. This button opens a window like the one shown in Fig. 3.25 to allow the user to load a JSON file with the topology. The format of the JSON file is the one described in Fig. 3.11. Like what happened in the last interface, if the topology file contains errors, those will be prompted in the textbox.

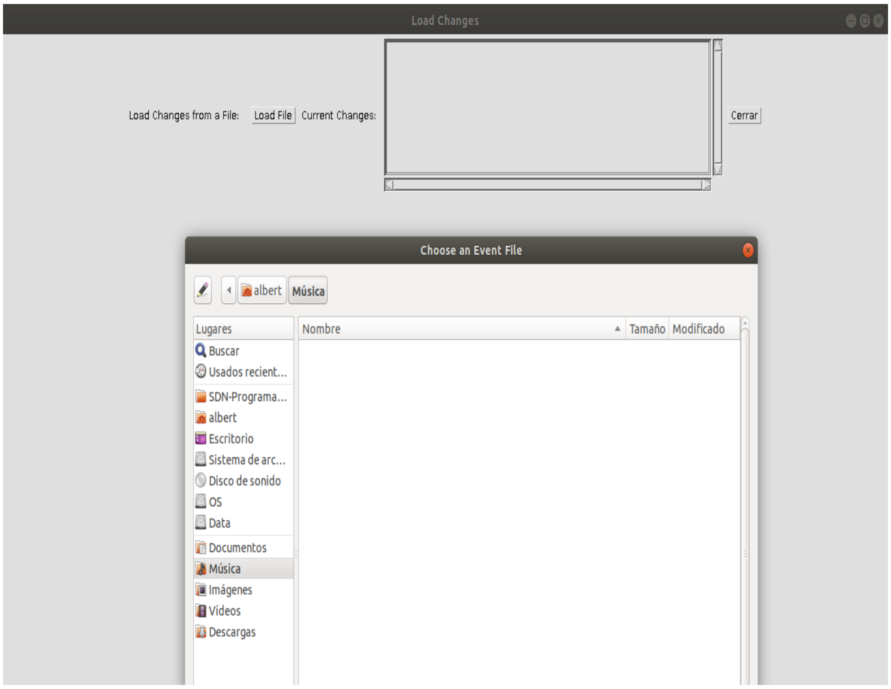


Figure 3.25: Load File window.

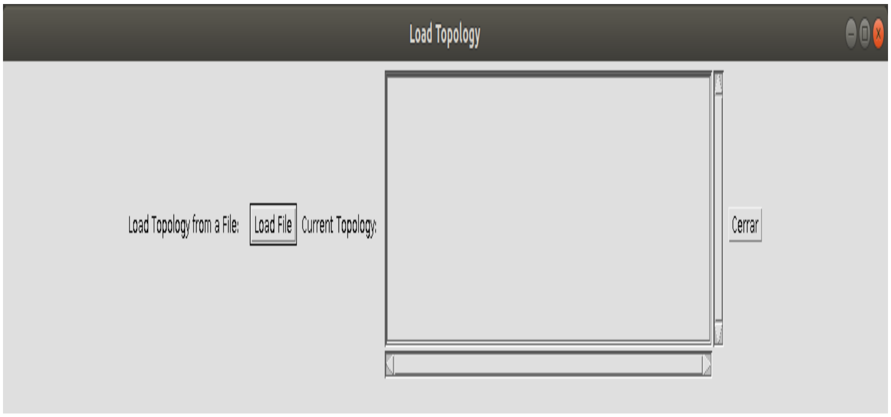


Figure 3.26: Load Topology interface.

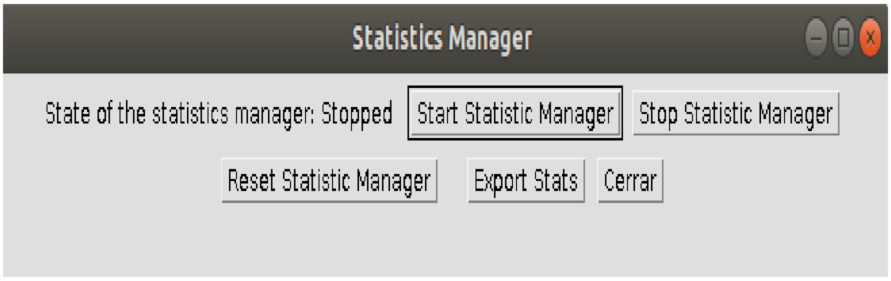


Figure 3.27: Statistics Manager interface.

The last interface accessible through the main interface is the Statistic Manager. This user interface, which is shown in Fig. 3.27, can be used to change the state of the statistic gathering. At the beginning of the execution, the SDN controller does not collect any data from the nodes, except the one needed to work. When the user presses the "Start Statistic Manager", the SDN controller starts running and it periodically gathers statistics from all the OpenFlow-enabled nodes in the network. The user can stop the statistic gathering or even reset it, to erase all the data the SDN controller had collected so far. The last button is used to export the data in several .csv files to be analyzed by the user.

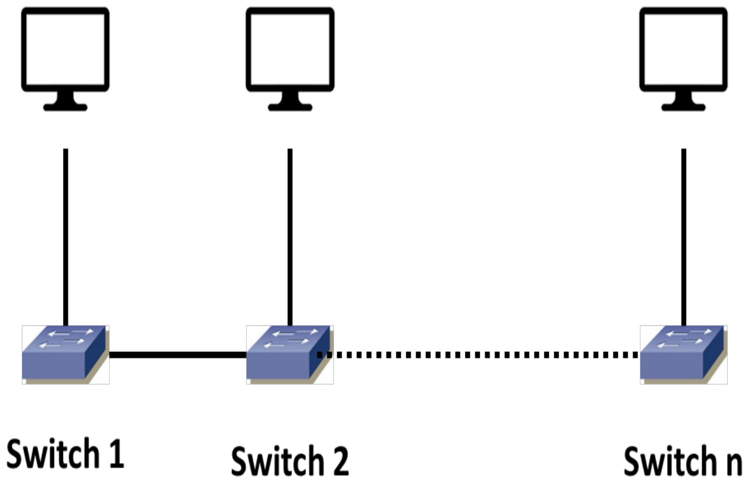


Figure 3.28: Linear Topology.

3.8 Performance Evaluation

In this section, a performance evaluation of the SDN controller is presented. This evaluation is done based on the resources consumed by the SDN controller, in terms of RAM memory and CPU.

The SDN controller is tested against Floodlight performance. In order to do that, several networks have been emulated in Mininet. The emulation is run in an Asus laptop with a CPU Intel Core i7-55000U 2.40GHz CPU and 8GB of RAM. The networks emulated correspond to different topologies. In each topology, the size is increased. For each size, the percentage of CPU and RAM used by the controlled is obtained through a 1-minute-long execution. The minimum, average and maximum values are compared in each scenario. Next, the different topologies and the results obtained in each one of them are discussed.

3.8.1 Linear Topology

The linear topology is the simplest one. It consists of connecting the elements of the network one after other. Fig. 3.28 depicts the structure of this topology.

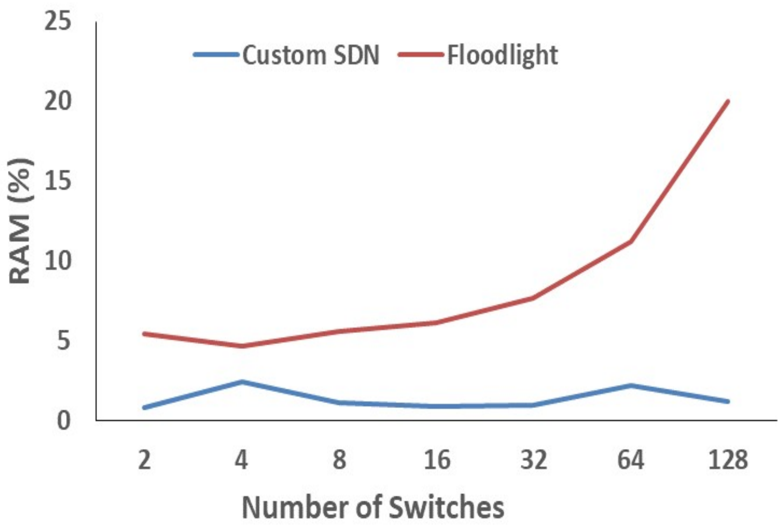


Figure 3.29: Minimum RAM used in linear topology.

The Mininet emulator allows setting the number of switches in the network. Let s be the number of chosen switches in the network, the number of hosts and links are defined by equations (3.1) and (3.2), respectively.

$$H = S \tag{3.1}$$

$$L = 2S - 1 \tag{3.2}$$

In the experiment, the number of switches increased from 2 to 128. Fig. 3.29 shows the evolution of the minimum percentage of RAM used by both controllers. The results obtained show that Floodlight has a greater minimum RAM usage than the custom SDN controller. Furthermore, this minimum RAM is increasing depending on the number of switches in the network, which also determines the size of the network in terms of hosts and links used. With a number of switches from 2 to 8, the minimum percentage of RAM used by the custom SDN controller varies from 1 to 2.4%. Floodlight presents values from 4.7 to 6%. This difference is increased when the number of switches is from 32 to 128. In this case, the custom SDN controller presents minimum values around the 2% and Floodlight from 7.7 to 20%.

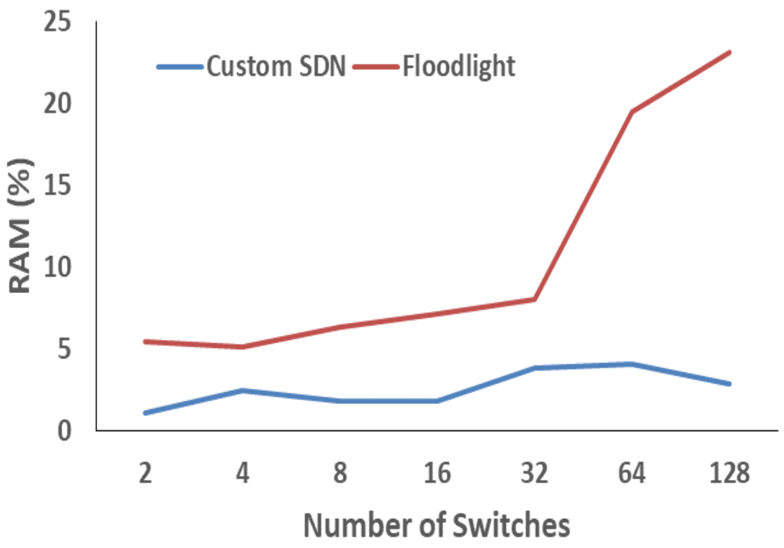


Figure 3.30: Average RAM used in linear topology.

Fig. 3.30 depicts the average percentage of RAM used by both controllers depending on the number of switches. The graphic shows that Floodlight consumes more RAM than the custom SDN controller. Once again, the difference is greater with more switches in the network. With a low number of switches (2-16), the Floodlight controller wastes an average of 5.12-7.13%, while the custom SDN controller uses an average of 1.05-2.4%. When the number of switches is higher, from 32 to 128, Floodlight presents an average of 8-23% and the custom SDN controller around 4%.

As regards the maximum RAM used by the controller, Fig. 3.31 shows that there are not big differences with the previous results. With a low number of switches, the custom SDN controller wastes a maximum RAM percentage of 1.1-2.4% while Floodlight from 5.3% up to 7.1%. On the other hand, with the number of switches from 32 to 128, the maximum RAM used by Floodlight increases to 8.1-25.4% and the maximum RAM used by custom SDN up to 3.2-5.1%.

Fig. 3.32 shows the evolution of the minimum percentage of CPU used by both controllers. While Floodlight consumes quite a constant minimum quantity of CPU (from 2.88-3.48%), the custom SDN controller presents an increment of minimum CPU consumption when the number of switches increases. The controller consumes a minimum CPU of 1-3.97% with 2-32 switches. Neverthe-

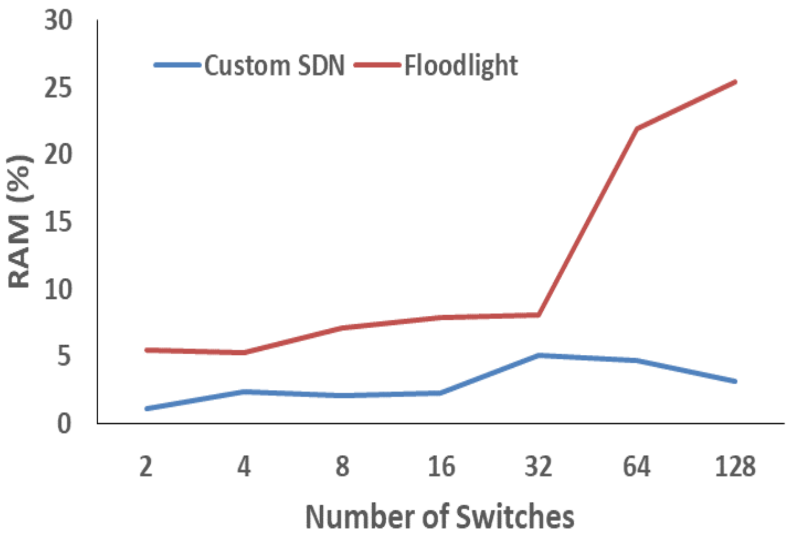


Figure 3.31: Maximum RAM used in linear topology.

less, this minimum quantity is increased up to 9.11-16% with 64-128 switches. These results show a more constant initialization from Floodlight.

In terms of average CPU consumed, Fig. 3.33 depicts the evolution of this consumption presented by the controllers. Both controllers show an increment of the CPU consumed along with the number of switches. Floodlight consumes an average of 3% with 2-8 switches and 4.5-17.2% with 16-128 switches. Custom SDN shows a more irregular increase. With a low number of switches, 2-8, the controller consumes an average of 3.9-6%. However, with 16-128 switches, the custom SDN controller consumes an average CPU between 14.1% and 35.32%, higher numbers than the ones presented by Floodlight.

The maximum CPU used with different number of switches in the linear topology is depicted in Fig. 3.34. The maximum CPU consumed by Floodlight remains in low levels, around 3.5%, from 2 to 8 switches. When the number of switches is increased up to 16, the maximum CPU values increases too up to 11.1%. And when it is increased up to 128, the maximum increases up to 33.68%. On the other hand, the custom SDN controller shows an earlier increment of maximum CPU usage. With 2 and 4 switches, the maximum value is 5.3% and 4%, respectively. However, with 8 switches is 11.32%. With 16-128 switches, the maximum percentage of CPU used is from 36.13% to 39.41%.

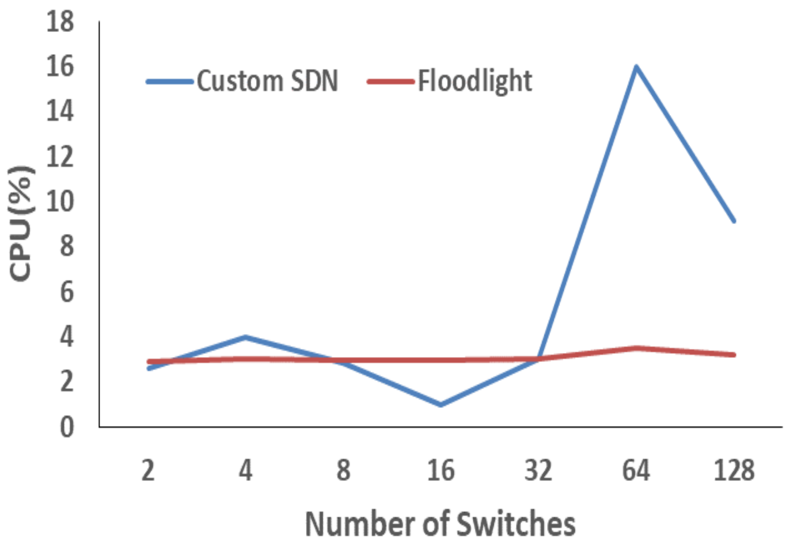


Figure 3.32: Minimum CPU used in linear topology.

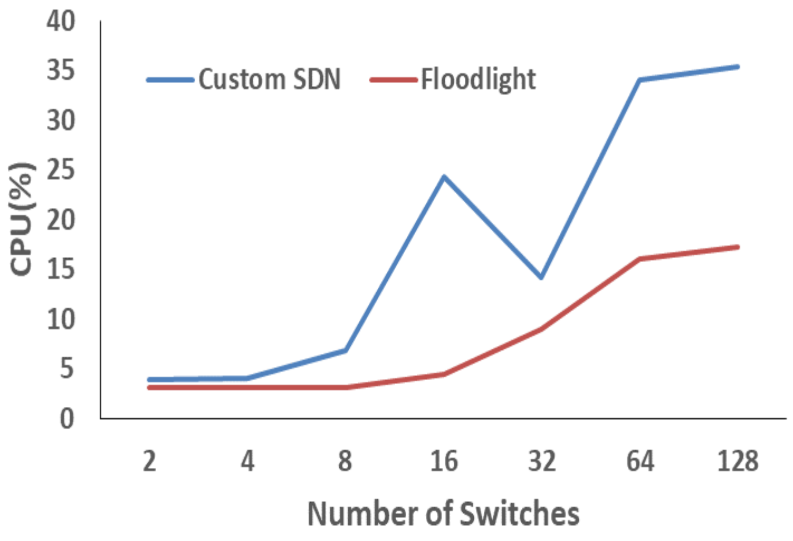


Figure 3.33: Average CPU used in linear topology.

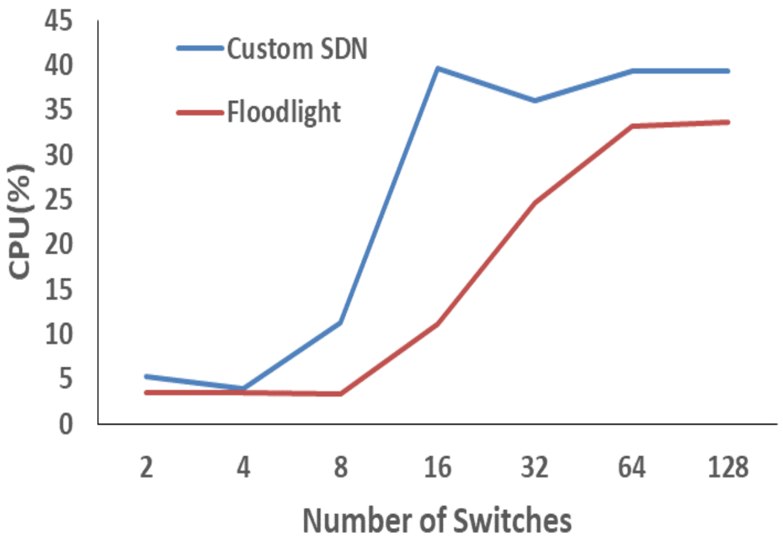


Figure 3.34: Maximum CPU used in linear topology.

3.8.2 Tree Topology

The next tested topology is the tree topology. In this topology, the nodes of the network are arranged like the branches of a tree. Mininet introduces a tree topology like the one shown Fig. 3.35. The last switches of the tree are connected to the hosts.

The Mininet emulator allows setting the depth of the tree. It allows setting the width too, but in this work, the width will be 2, the minimum, and the depth will vary. Depending on how deep the network is, the number of nodes interconnected will increase. Let d be the depth of the tree. The number of switches, hosts and links are defined by equations (3.3), (3.4) and (3.5), respectively.

$$S = 2^d - 1 \tag{3.3}$$

$$H = 2^d \tag{3.4}$$

$$L = 2S \tag{3.5}$$

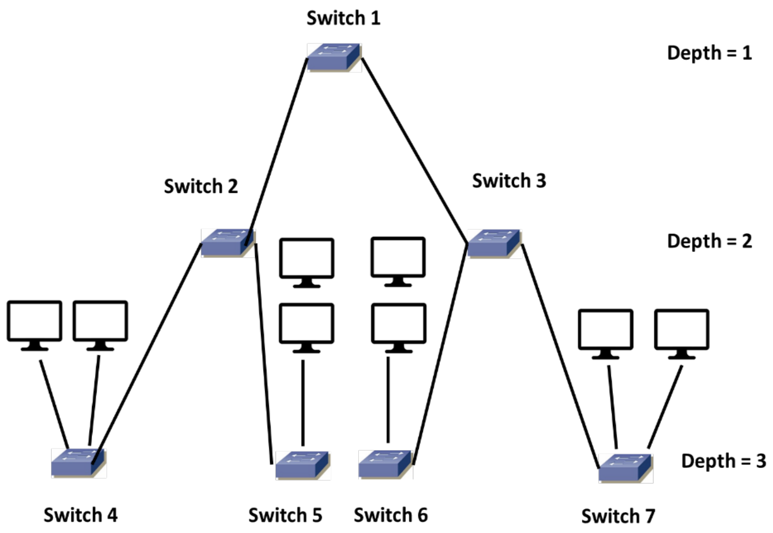


Figure 3.35: Tree Topology.

For the experiments, the depth of the tree was varied from 2, the minimum accepted by Mininet, to 7. Again, the percentages of CPU and RAM used, minimum, average, and maximum, were gathered. Fig. 3.36 shows the minimum RAM used, in percentage, by the controllers. The custom SDN controller achieves a good minimum usage of RAM between 1.1% and 2.1%. Floodlight, on the other hand, presents a significant increment of the minimum RAM used. With a tree depth of 2-4, Floodlight consumes a minimum RAM percentage of 4.2-6.2%, while with a tree depth of 5-7, this consumption is increased to 7.3-19.4%.

Fig. 3.37 depicts the average RAM used during the execution by both controllers. This case is quite similar to the last one. The custom SDN controller has a lower RAM consumption, although, this time, it increases more its consume when the depth of the tree increases too. The values for a depth of 2-4 are around 1.8%, while with 5-7 they go from 3.1% to 5.15%. The increment of these values is higher with Floodlight, consuming between 4.3% to 6.6% for depths of 2-4 and 8.5% to 24.07% from 5-7.

The maximum RAM used by the controllers presents a similar tendency to the average RAM used. This can be appreciated in Fig. 3.38. In the maximum percentage of RAM, the custom SDN controller starts with a consumption of 1.3-2.3% for depths from 2 to 4. This consumption is raised up to 3.5-7%

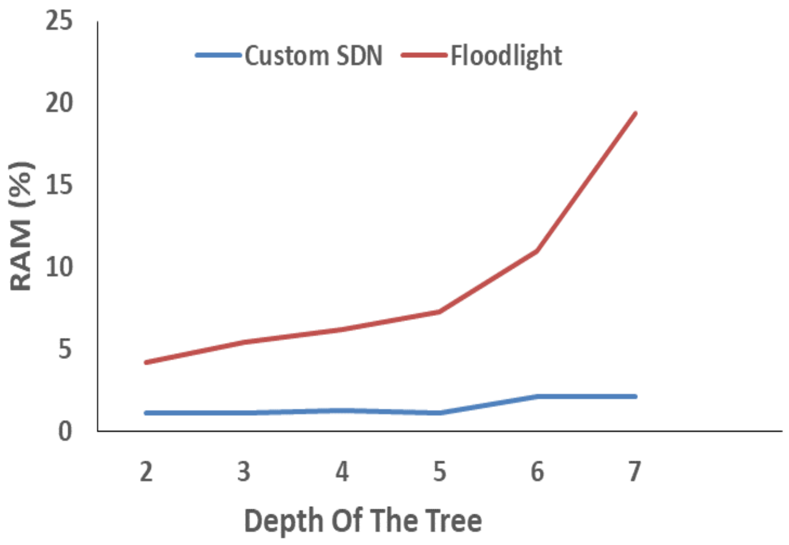


Figure 3.36: Minimum RAM used in tree topology.

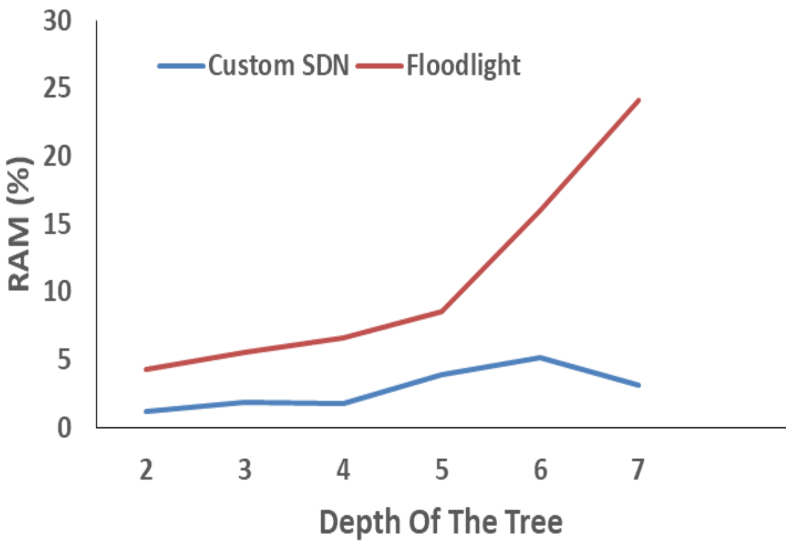


Figure 3.37: Average RAM used in tree topology.

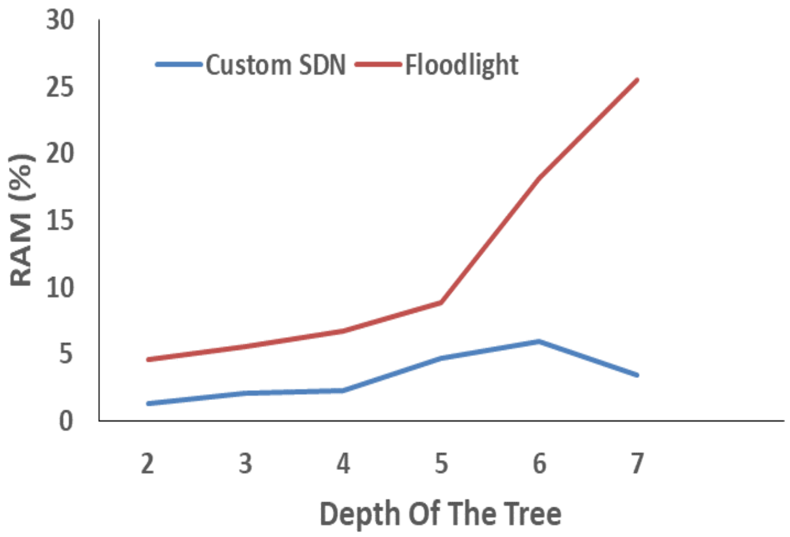


Figure 3.38: Maximum RAM used in tree topology.

for depths from 5 to 7. Floodlight shows again a remarkable increment of consumption, with percentages of 4.6-6.8% for depths from 2 to 4 and 8.9-25.5% for depths of 5 to 7.

As regards CPU, Fig. 3.39 shows the minimum CPU percentage used by both controllers. In this case, both controllers show a similar consumption. The custom SDN controller shows minimum CPU consumptions of 0.3-2.76% for depth values of 2-4 and consumptions of 0.3-15.12% when the depth of the tree is from 5 to 7. The values from Floodlight are 3.08-3.28% when the depth is from 2 to 4 and 3.01-13.43% for depths of 5-7.

More interesting are the values of the average CPU consumptions, shown in Fig. 3.40. In this case, although both controllers present an increasing tendency, the values of the custom SDN controller are higher. These values are 19.75-23.86% for depths from 2 to 4 up to 17-33.87% with depths of 5-7. The values from Floodlight are 3.16-4.47% when the depth is from 2 to 4 and 5.31-23.6% for depths from 5 to 7.

Finally, in Fig. 3.41, the maximum CPU consumption values are shown. The graphic shows that the Custom SDN controller presents higher CPU consumption peaks. These are quite constant, between 34.46% and 39.7% during all the executions. Nonetheless, Floodlight presents lower maximum CPU values

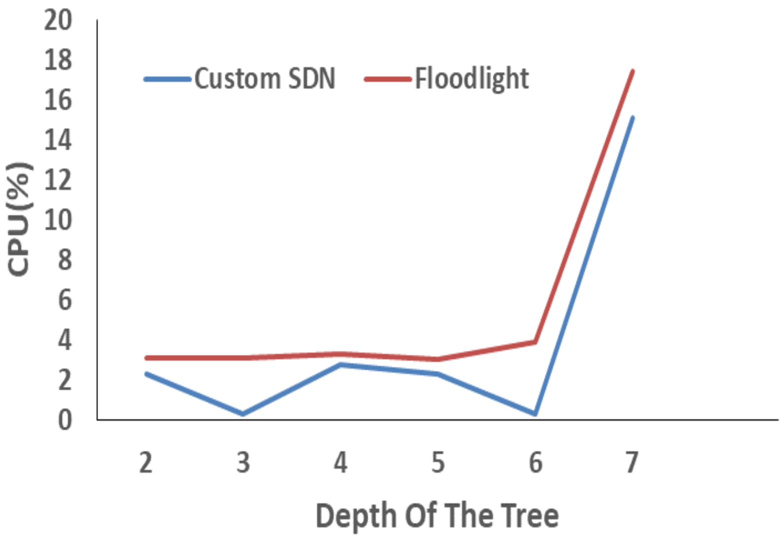


Figure 3.39: Minimum CPU used in tree topology.

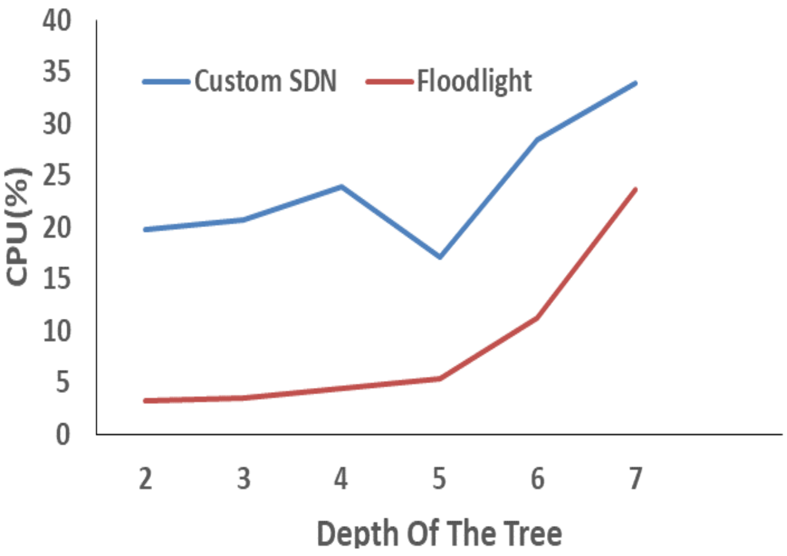


Figure 3.40: Average CPU used in tree topology.

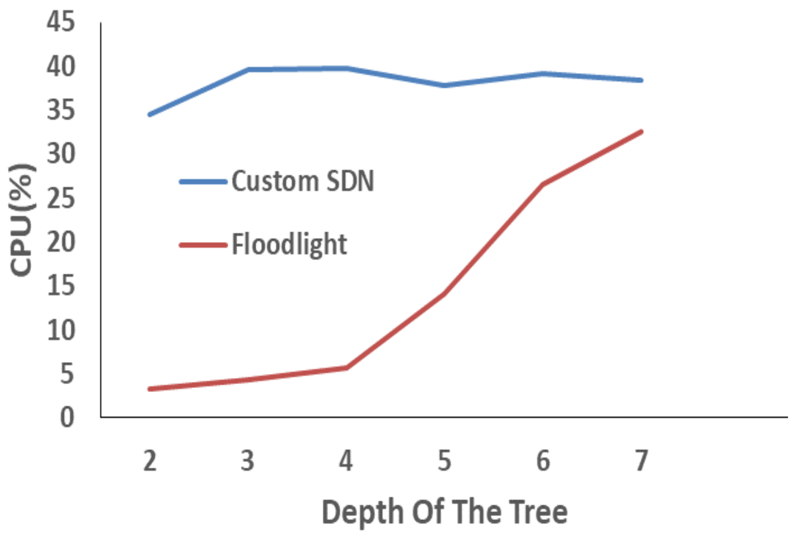


Figure 3.41: Maximum CPU used in tree topology.

with depths from 2 to 4, from 3.28% to 5.62%. The values for depths from 5 to 7 are between 14.03% and 32.57%.

3.8.3 Torus Topology

The last topology tested is the torus topology. In this topology, the network nodes are connected forming a mesh. Fig. 3.42 shows an example of this kind of topology. Each switch is connected to a host. The dotted or scattered lines represent connections between the edge nodes of a row or a column.

This topology presents a high redundancy, interconnecting the different layers of the mesh. The Mininet emulator allows setting the size of the mesh. Let m be the size of the mesh, the number of hosts, switches and links are defined by equations (3.6), (3.7) and (3.8), respectively.

$$S = m^2 \tag{3.6}$$

$$H = m^2 \tag{3.7}$$

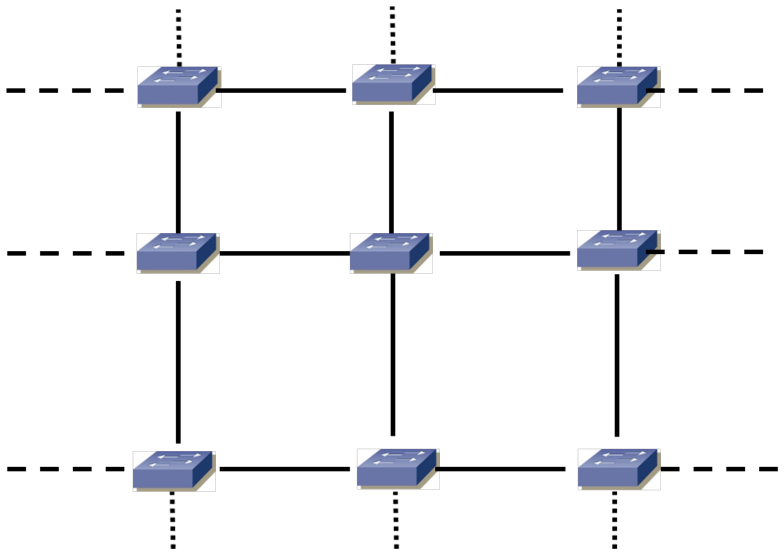


Figure 3.42: Torus Topology.

$$L = 3m^2 \tag{3.8}$$

In the test, the size of the mesh was increased from 3 to 9. Again, the values of minimum, average, and maximum CPU and RAM consumptions were compared. Fig. 3.43 depicts the minimum RAM used during the execution by both controllers. Custom SDN shows constant minimum values regardless the size of the mesh. Its values fluctuate between 2.1 and 2.7%. However, Floodlight presents a minimum RAM usage between 6.2% and 19.42%, varying significantly depending on the size of the mesh.

Fig. 3.44 shows the average RAM used. Both controllers present different average RAM usage with the increment of the size of the network. Nevertheless, Floodlight increases it when the size of the network increases. It starts with 6.2-18.18% with a mesh of 3-5 layers and increases up to 12.1-19.425% with a mesh size of 6-9. The custom SDN controller, however, presents its highest average RAM usage with a size of 3 (10%) and decreases, 2.7-5.76% with a size of 4-6 and 2.175-5.575% for sizes from 6 to 9.

The behavior of the average RAM consumption is also found in the maximum values of RAM usage. In Fig. 3.45, those values are depicted. Floodlight increases again the maximum RAM consumption from 6.2% to 18.1% with

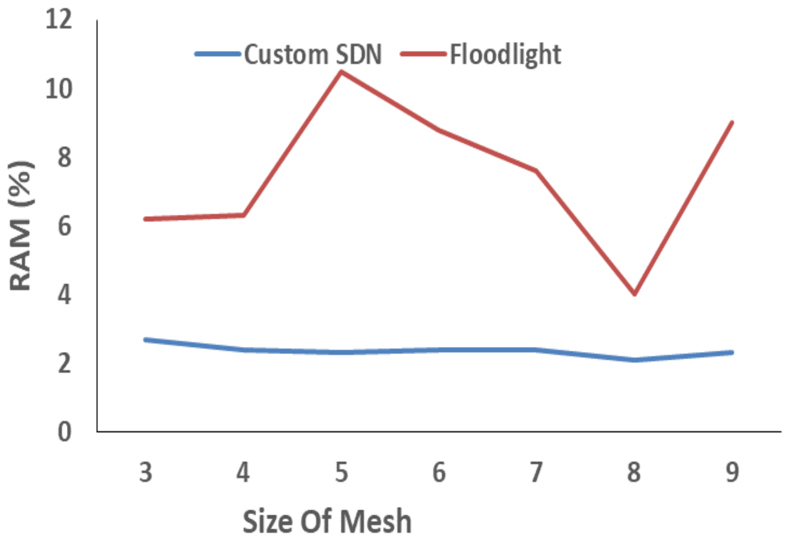


Figure 3.43: Minimum RAM used in torus topology.

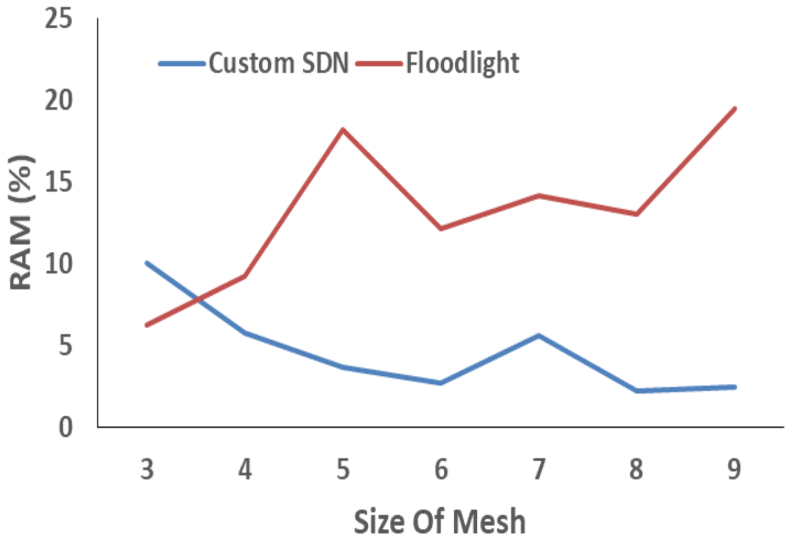


Figure 3.44: Average RAM used in torus topology.

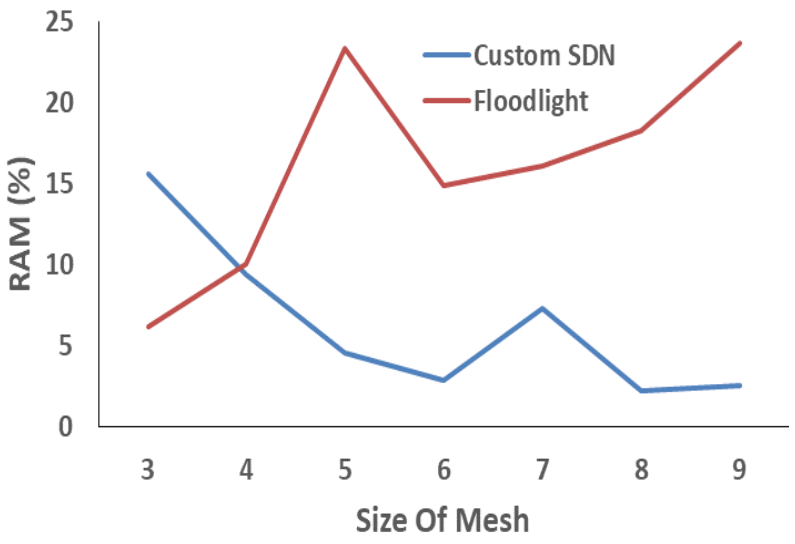


Figure 3.45: Maximum RAM used in torus topology.

lower sizes, from 3 to 6, up to 13-19.425% for sizes from 6 to 9. The custom SDN controller presents higher peaks of RAM usage with low sizes too, from 2.9% to 15.6%, for sizes from 3 to 6, up to 2.2-7.3% for higher sizes, from 7 to 9.

In terms of CPU usage, the next figures show the differences between the controllers. Fig. 3.46 shows the minimum percentage of CPU used by the controllers. Floodlight presents lower minimum values, especially with small sizes of mesh, from 3 to 5, when Floodlight consumes 2.74-6.79%. With sizes from 6 to 9, the CPU minimum consumption increases from 4.59% to 24.54%. The custom SDN controller presents a minimum CPU consumption of 13.38-32.35% for sizes from 3 to 5 and 28.95-33.89% for sizes from 6 to 9.

The average CPU used during the execution by each SDN controller is depicted in Fig. 3.47. Floodlight shows lower average CPU consumption values than the custom SDN controller. Floodlight also presents an increment, from 7.2-18.97% with sizes of 3-5 up to 24.64-30.6% for sizes of 7-9. On the other hand, the custom SDN controller has a more constant CPU usage, regardless the size of the mess, with consumptions from 32% to 36.85%.

Finally, Fig. 3.48 represents the maximum peaks of CPU usage for both SDN controllers. The values are similar for both controllers. Floodlight has peaks

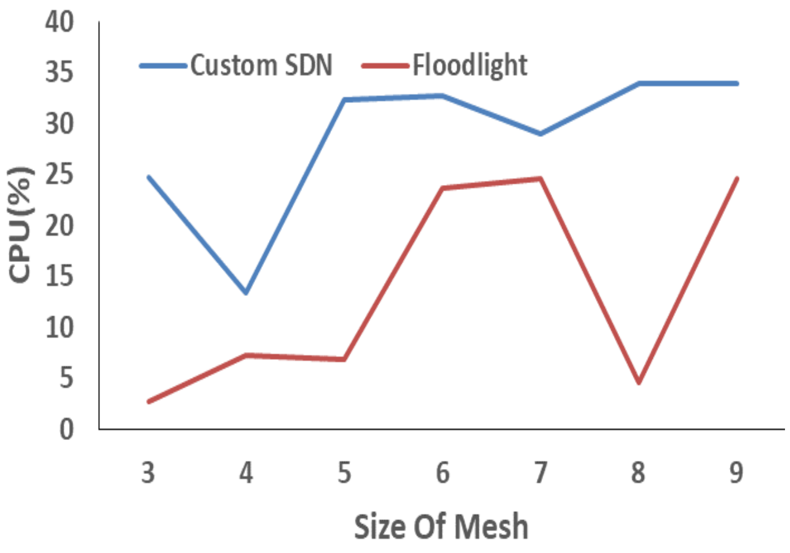


Figure 3.46: Minimum CPU used in torus topology.

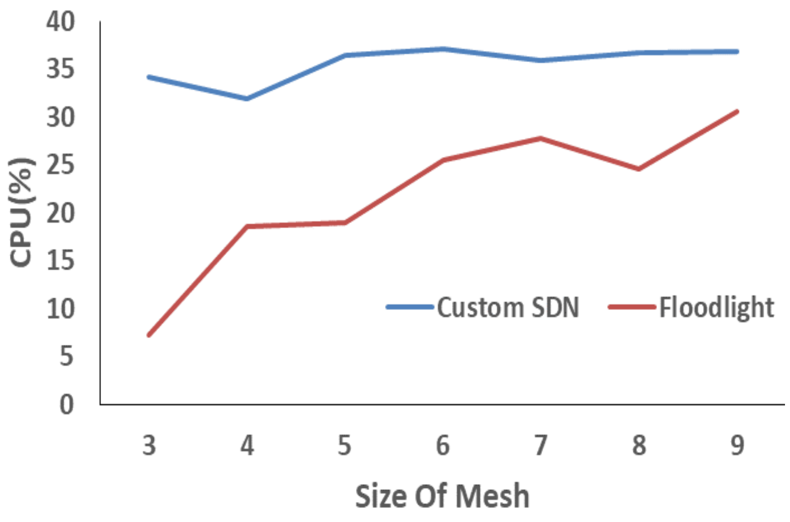


Figure 3.47: Average CPU used in torus topology.

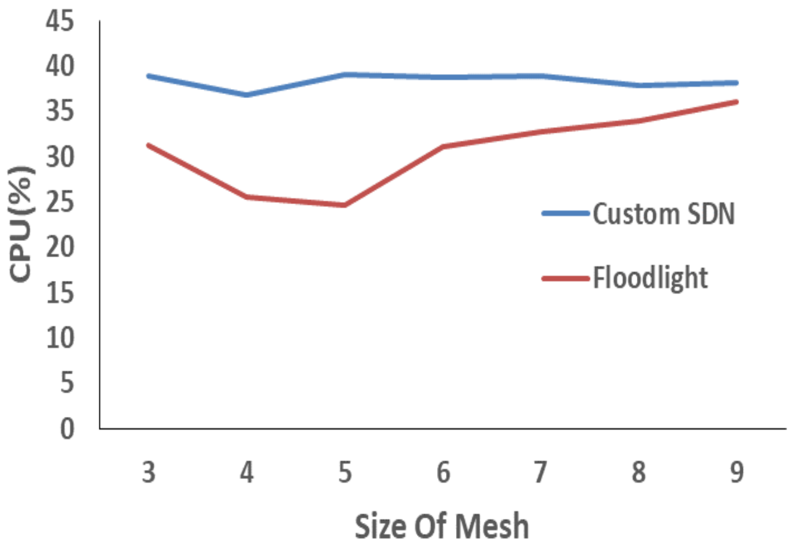


Figure 3.48: Maximum CPU used in torus topology.

of consumption of 24.62-31.2% for sizes from 3 to 5 and peaks of 31.11-36% for sizes from 6 to 9. The custom SDN controller presents maximum values of 36.8-38.85%.

3.9 Conclusion

In this chapter, the following points have been addressed:

- A SDN Background has been discussed. This helps the reader to understand the technical details of SDN and Openflow. Thanks to that, the next points of the dissertation can be easily understood.
- The SDN controller has been introduced. First, the general goal of this SDN controller and its architecture has been discussed. Then, the design details have been shown: user cases, class diagram, sequence diagrams and code fragments.
- The topology discovery process has been discussed. Moreover, the GUI of the SDN Controller have been shown and described.

- A performance evaluation of the SDN controller has been shown so as to demonstrate the capabilities of the controller in terms of CPU and RAM consumption.

The SDN controller developed and described in this chapter is the base from all the rest of the proposal is built on. Therefore, all the proposals described in the next chapters uses the SDN controller software detailed in this chapter.

Chapter 4

Routing Proposal

4.1 Introduction

In this chapter, the routing protocol that implements the solution of this work is proposed. The chapter consists of two main sections. On the one hand, a modification of the well-known OSPF routing protocol is discussed. This modification takes advantages of the capabilities of SDN to improve the multimedia transmission. On the other hand, a completely novel routing protocol is proposed as the final solution. This protocol is designed from scratch taking into account the possibilities of SDN technology.

However, first we need to introduce the architecture of the SDN network. Since the architecture used is SDN-based, the routers are SDN nodes, i.e. OpenFlow-enabled devices. Besides, there is a central controller, whose aim is to manage the entire network, obtaining the current state of the network and making orders to the nodes. To this usual SDN architecture, an AI module is added. It is in constant communication process with the controller and it is used in order to get the cost of the different possible paths in routing decisions. This architecture is shown in Fig. 4.1. On the left side, the difference between the controller and switches is shown, and their interaction using OpenFlow is displayed. On the right side, the network architecture shows the connection between the different actors in the SDN architecture.

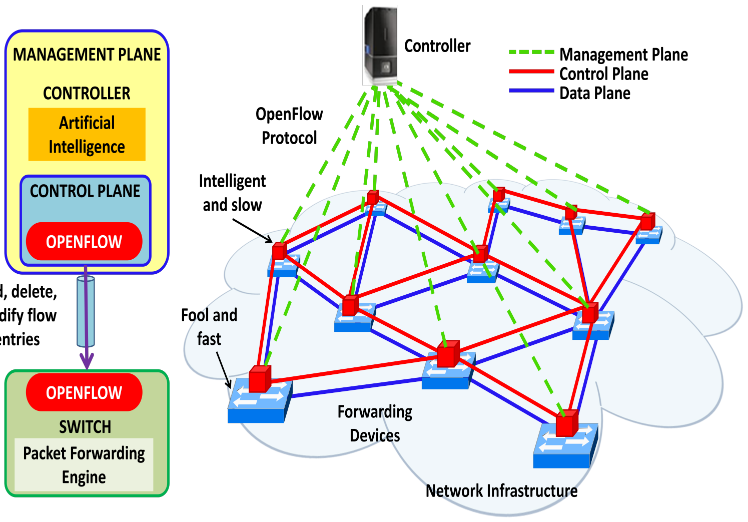


Figure 4.1: Elements and network of the proposed architecture.

4.2 Dynamic OSPF Adaptation

In this section, the proposal of adapting OSPF to SDN multimedia transmission is detailed. First, the proposed routing protocol is described. Later, the messages are shown. Finally, the algorithm and messages exchanging process are discussed. Finally, an evaluation of the proposal is provided.

4.2.1 Routing Proposal

Routing algorithms can be depicted in several blocks attending to the different functions. Those blocks are shown in Fig. 4.2. The first module, painted in purple, is the one that builds the routing tables. OSPF is based on link state algorithm. The second one is the group of messages and communication process that provides the possibility to the protocol to work. The messages are exchanged between the routers. The green one contains the metric calculation. Depending on the protocol, the metric calculation can slightly vary. However, there is always a determined formula to calculate the cost of the links. Finally, the protocol has other determined process and functions to bring some new working ways. For instance, OSPF can manage administrative areas.

Our proposal modifies two of those four modules, changing and expanding their function. The new structure is detailed in Fig. 4.3. The figure is quite similar to the one explaining the structure of OSPF. The same four modules (Routing Table, Messages and Communication, Metrics Calculation and Other Protocol Functions) are defined. However, as can be observed, some of these modules have been modified. The routing table creation algorithm has been split into two different modules that offer the same function. The link state algorithm has not been modified, but a new module called "Link State Dynamic Checking" has been added to the route table creation. This module has been introduced thanks to the possibilities given by the SDN. The nodes are able to gather the statistics of the flows sent through the different paths and send them to the controller. So it allows the SDN routing protocols to make use of actualized link state data. It increases the possibility of adapting the routing protocol to the current state of the network and the changes produced in the network.

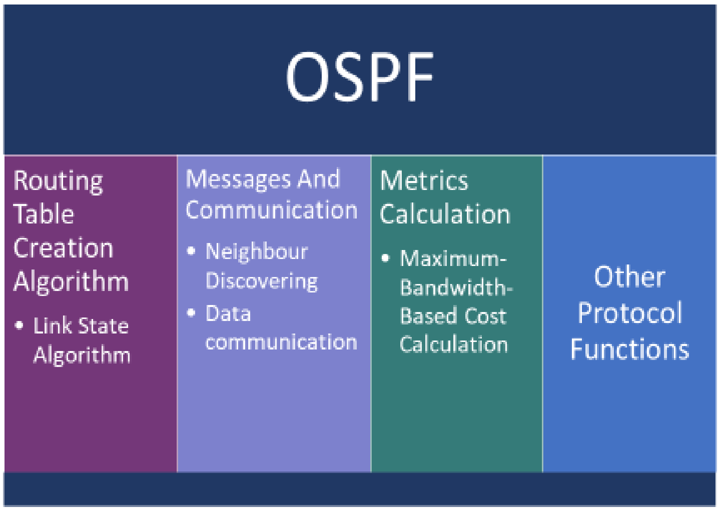


Figure 4.2: Routing protocol structure.

This possibility is quite important to choose the best path for all different kinds of traffic, but especially for multimedia traffic, due to the possibility to not only to avoid links down or links state changes. In addition, it can be used to get the evolution of the links utilization and be able to apply decisions to ensure minimum QoS (and QoE) levels. These levels show the quality of the multimedia communications.

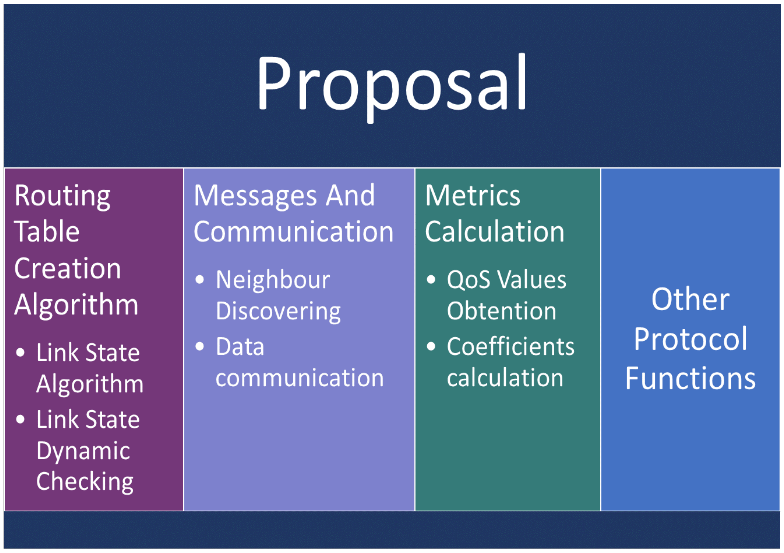


Figure 4.3: New routing protocol structure.

The other important change introduced by the proposal is the modification of the metric calculation. OSPF uses the formula shown in (4.1) to calculate the cost of each link.

$$Cost = \frac{10^8}{Bandwidth(bps)} \tag{4.1}$$

OSPF uses the bandwidth of the links to calculate the cost. However, when the protocol is especially designed for multimedia traffic, there are some other variables that must be considered. Those variables are the QoS factors, which are closely related with the quality that the user perceives. In (4.2), the EIGRP metric formula is shown [143].

$$Cost = [K1 * BW + \frac{K2 * BW}{256 - load} + (K3 * delay)] * \frac{K5}{reliability + K4} \tag{4.2}$$

It uses several QoS factors like bandwidth, calculated as in (4.3), delay and loss rate. The coefficients K1-K5 are used to take into account the factors or not.

$$BW = \frac{10^8}{Bandwidth(bps)} \tag{4.3}$$

Usually, the EIGRP metric is calculated as in (4.4), taking 1 as the value of the coefficients K1 and K3 and 0 for the others. Therefore, K1-K5 coefficients act as weights for each QoS factors. EIGRP uses only bandwidth and delay as QoS factors.

$$Cost = BW + delay \tag{4.4}$$

Our proposal also considers different QoS factors, calculating the cost as in (4.5).

$$Cost = \frac{K1 * 10^8}{C.Bandwidth(bps)} + K2 * Delay + K3 * losses \tag{4.5}$$

This equation has been designed to take into account the main QoS factors, due to the main role played by the multimedia traffic in the proposal.

The equation is composed of two main elements: a set of QoS factors (bandwidth in bps, delay in seconds and loss rate in percentage) and another one of constant values (K1, K2 and K3). The calculation of these two factors are quite different and it is indicated in Fig. 4.3, separated in two different modules.

On the one hand, we have the QoS factors. In the OSPF equation, the bandwidth is a constant, it is the maximum quantity of data that can be sent per second through a specific link when the route table is established. This value does not change unless a topology change happens. In the proposal, there are two changes from the OSPF metric calculation. First, the bandwidth, just as the other QoS factors, it is time-depending. Its values change along the time. They are based on the statistics provided by the nodes and the controller. These values are dynamically changing, an important fact to maintain QoS and QoE levels in multimedia communications. The other difference is the meaning of the factors. Bandwidth, for being calculated in a dynamical way, is based on the current bandwidth, not the capability of the link. This allows applying techniques like load balancing to avoid a QoE falling, making a better management of the network resources. This is possible because of the use of SDN and the statistics gathering of the nodes in every flow that goes through each link.

On the other hand, regarding the coefficients, they are used to increase the weight of some factors depending on the main traffic in the network. Observing the formula, it seems that bandwidth can be the most important QoS factor because it affects in the greatest way to the cost. But it depends on the coefficient's values. For example, if the network is being used for video streaming under demand, the importance of the available bandwidth is really greater than the delay. So, K1 should be greater than K2. Otherwise, if the main traffic in the network is multimedia streaming, the weight of the different factors should be similar. Finally, the network could be used for VoIP, where the delay is a crucial factor. In that case, K2 should be greater than K1. The fact of varying the method used to calculate the path cost can improve the performance of the network. This variation is produced by the AI module, in the controller. The AI module will set different values to the coefficients in order to get the best performance in terms of QoS. The SDN controller will manage the different kind of traffic associating them to some categories. This will allow the controller to label the flows and be aware of the main traffic in the network. There are two ways of working with these categories. On the one hand, the category related to the greatest quantity of flows can determine the values of the metric factors. On the other hand, for each flow the system can use the metric values related to the category it has been labeled to.

In conclusion, SDN brings us the possibility to modify the routing algorithm to get dynamically the cost of each path, being able to change the metric formula depending on the state of the network. To achieve this, it is important to be able to use some structure like the flow table that associates the different flows in the network with the kind of traffic that is composed by. Moreover, it is necessary to be able to have in memory the factors of the different categories for avoiding re-sending messages when the category used in the network changes.

4.2.2 Metric Analysis

Once the formula of the metric proposed has been discussed, an analysis of this metric is described in this subsection. First, the aim of the system is that the coefficients are determined by the AI module, during a learning process. However, the design of the equation is a result from a specifically chosen balance between its factors that is described in this subsection.

In the first place, the available bandwidth must be placed as a denominator and the delay and loss rate as nominators. The use of the Dijkstra algorithm and the cost as a metric to be measured, implies that the greater the cost, the worse the path is. So, the bandwidth must reduce the cost of the path.

Regarding the delay and the loss, the decision is that they affect to the cost in a similar way to the bandwidth in the initial point. Then, the AI module will change the coefficient values and will adapt the metrics to the different categories. Thus, an analytical method has been performed to determine an acceptable set of coefficient values. The values chosen for the coefficients are K1=1, K2=0.5 and K3=5. Fig. 4.4, Fig. 4.5 and Fig. 4.6 show the possible values of the cost for different bandwidth, loss rate and delay.

Fig. 4.4 shows the values of the metric when the links of the topology have a bandwidth of 10 Mbps. The different loss rate tested form different lines in the graph. From the blue line, with 1% of loss rate, to the yellow one, with 10%, the cost obtained with the formula is displayed for different delay values. The delay values increase from 10ms to 3000ms. With the minimum delay, the costs obtained are 15.5, 25.5, 35.5 and 60.5 for 1%, 3%, 5% and 10% loss rate respectively. With 1s of delay, the cost values are 65, 75, 85, and 110. Finally, with 3s of delay the values are 165, 175, 185 and 210.

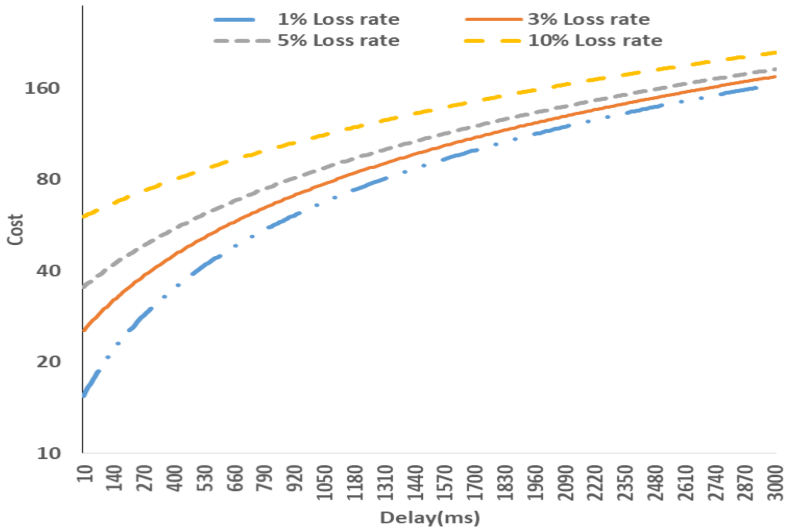


Figure 4.4: Cost per delay and loss rate with a bandwidth of 10Mbps.

Fig. 4.5 shows the values obtained for a topology with 100 Mbps of available bandwidth. The increments of delay and loss rate are the same than in the previous graph. With the minimum delay, we have 15.5 of cost when there is 1% of loss rate. With 3% of loss rate, the cost is 16.5, 10 points more than with 1% of loss rate. Finally, with 5% and 10% of loss rate the cost values are

26.5 and 51.5 respectively. The increment of the bandwidth from 10 Mbps to 100 Mbps is balanced with a 2% increment of loss rate. With 1s of delay, cost values are 56, 66, 76 and 101. Finally, with 3s of delay, cost values are 156, 166, 176 and 201. And with 210ms, the cost with 1% of loss rate increases from 6.5 to 16.5. So, the increment of 200ms of delay is equivalent to the increment of 2% of loss rate.

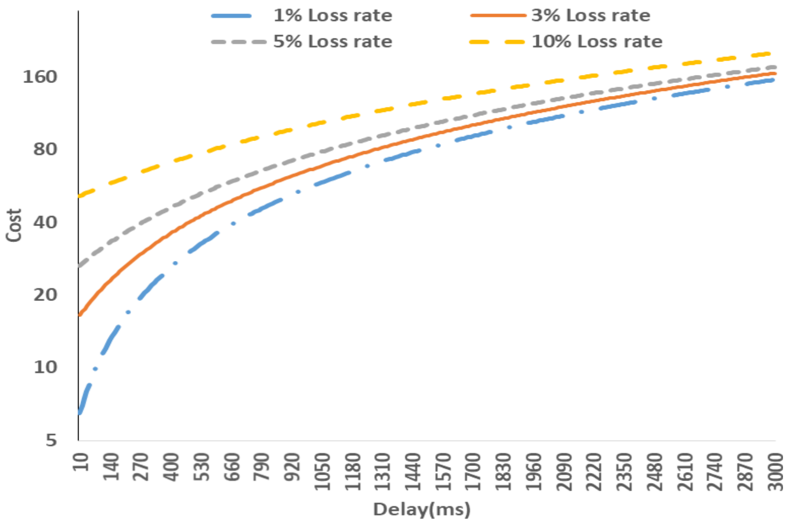


Figure 4.5: Cost per delay and loss rate with a bandwidth of 100Mbps.

Fig. 4.6 displays the cost with 1000 Mbps. The minimum cost is reduced to 5.6 when there is 1% of loss rate. This is a reduction of only 0.9 points. This is caused when there are different paths to choose with high bandwidth. The differences between these bandwidths are not relevant, and the other QoS factor reduction must be taken into account. These QoS factors, delay and loss rate must be the differential factor in these cases. The other cost values with minimum delay are 15.6, 25.6 and 50.6 with 3%, 5% and 10% respectively. With the minimum delay, the costs are 15.5, 25.5, 35.5 and 60.5 for 1%, 3%, 5% and 10% loss rate respectively. With 1s of delay, the cost values are 55.1, 65.1, 75.1 and 100.1. Finally, with 3s of delay the cost values are 155.1, 165.1, 175.1 and 200.1.

These graphs demonstrate that, initially, the coefficient factors are balanced. Therefore, the network can work properly and the AI module will adjust the coefficients depending on the use of the network and changing between the

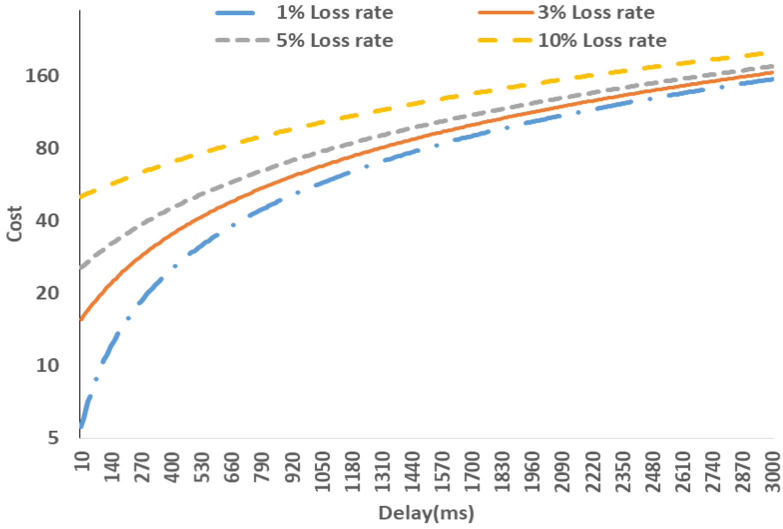


Figure 4.6: Cost per delay and loss rate with a bandwidth of 1000Mbps.

different categories. That is possible thanks to the messages described in the next subsection.

4.2.3 Messages

In Fig. 4.7, the structure of the messages exchanged between the SDN controller and the nodes is described.

The messages are detailed below:

- Categories_init: initialize the algorithm, indicating the number of traffic categories that is going to be used.
- Categories_use: notifies the category that is going to be used as a reference to calculate the cost of the path.
- Categories_factors: the factors used in a specific category. It is mainly used to actualize the values of the factors in a category. The category and the values of the three factors are indicated.
- Flow_label: used to assign a specific category to a flow. Useful to treat some flow in a specific way, which is optional and allows each kind of

traffic flow to be treated with a different metric, not with the category associated to the majority of the traffic. Due to this optional ability, the first byte of the Flow_ID field is zero if the flow will not be treated with its own category metric, which is the normal behavior. Otherwise, the first byte is set to ones to indicate that the flow must use the metric of its category, even the category used is another one.

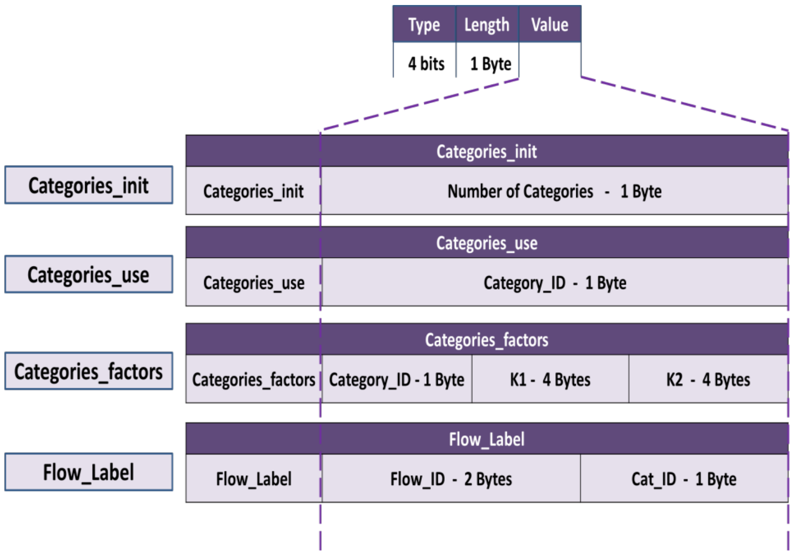


Figure 4.7: Messages structure.

4.2.4 Algorithm and Process

In this subsection, we describe the algorithm and the messages flow in order to manage the factors and the categories. Algorithm 4.1 describes the category management process used by the controller.

First, the algorithm initializes the structures needed to manage and identify the main kind of traffic in the network. Then, a Categories_init message is sent to communicate the number of categories that is going to be used. For each category, a Categories_factors message is sent. This message includes the initial factors values that will be used when that category is being treated. Moreover, if the initial category is not the first one in being sent, a Categories_use message is sent from the controller. Later, the AI module is prepared. It will start to change all the factors of the initial category to find the most accurate balance between them. This change is done every specific period of time and

Algorithm 4.1

```
Given: Categories, Cat_initial

Initialize_Categories()
Send Cat_Init()
Foreach Category in Categories do
    Send Categories_factors()
End For
If Cat_initial != DEFAULT do
    Send Categories_use()
End If
AI_Initialize(Cat_initial)
Cat_Prev = Cat_initial
Foreach new iteration
    If Cat_Current != Cat_Prev do
        AI_Change_Category(Cat_Current)
    End If
    AI_Get_Factors()
    Send Categories_factors()
    Cat_Prev = Cat_Current
End Foreach
```

attending to the performance statistics gathered by the controller using the OpenFlow messages.

The process is similar to the one presented in [135]. It is important that the controller informs the AI module which category is being used in the network. Thereby, the factor values calculation is a process associated which each category. In addition, the statistics are also passed to the AI module. The AI module returns the values that will be used to calculate the metrics for the routing protocol when the current category is used. Those values are indicated by a Categories_factors message.

The message exchange process is detailed in the following lines and shown in Fig. 4.8. First, the messages, which are sent when the algorithm is initialized, are described. A Categories_init message is always sent in the beginning. Then, for each category, the categories factors must be set by sending a Category_factors message. Those initial values have not been calculated in a learning process yet. They are only initial standard values of the factors. The last step in the process is optional. Usually, the first category is the category used when the traffic of the network is mixed, and the factors have similar values. In order to set another category from the beginning, a Categories_use can be sent.

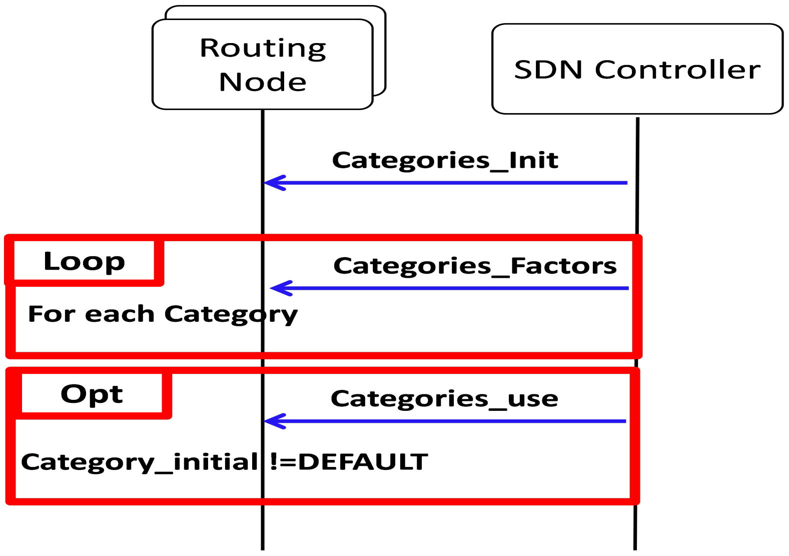


Figure 4.8: Initial message exchange.

The processes related to new flow and factors actualization scenarios are simpler than the initial process. When a new packet that is not classified is sent to a routing node, this node sends it to the controller by using the standard OpenFlow Packet_in message. Then, the controller analyzes the packet and classifies it into a specific category. It returns the packet and notifies that it has to be treated according to the routing protocol with the standard Packet_out message. The controller sends a Flow_label message to inform the node that the flow has to be classified as a specific category flow. This is useful if the optional way of working using different metrics simultaneously is desired. At that moment, the node can calculate the metric and send the packet from the adequate path. The controller can also send a Categories_use message if the majority of the traffic is now the one associated with a different category. This will depend on the current state of the network and the traffic that is flowing in that moment through the network.

Finally, the controller must communicate the values updated from the AI module to the nodes in order to allow them to work with the most updated version of the metric. For that, when there is an update of the values, the controller sends a Categories_factors message to the nodes. In that message, as it is explained in the previous section, the values used for one specific category are indicated. The controller will send one of those messages for each update. The controller could send a Flow_label message to change the behavior of some flow. This can be done to collect statistics and let the AI module learn.

Both processes described before are shown in Fig. 4.9 and Fig. 4.10 respectively. The messages exchanged between the controller and the routing nodes are shown. Moreover, the optional messages are also marked as “Opt”.

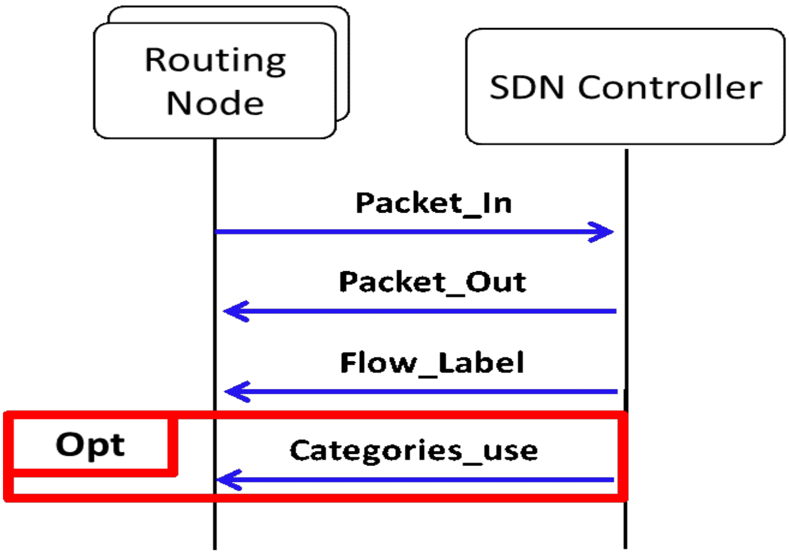


Figure 4.9: Message exchange when a new packet arrives

Furthermore, Fig. 4.11 shows a diagram as a graphical resume. In this diagram, the different processes and decisions are displayed. Each process described before can be identified in the diagram. The start of the algorithm is identified with the initialization of the categories, and the loop to inform the nodes about the new categories and the optional use of Categories_use message. Then, the main loop is reached. The factor update process and the new packet process that are shown in Fig. 4.9 and Fig. 4.10 can be identified as different branches of the main loop. The update provokes a new Categories_factors message to be sent and, optionally, a Flow_label message in order to continue learning. The new packet process consists of a Flow_label message and, if the main traffic of the network changes due to this new flow, a Categories_use packet is also sent to the routing nodes.

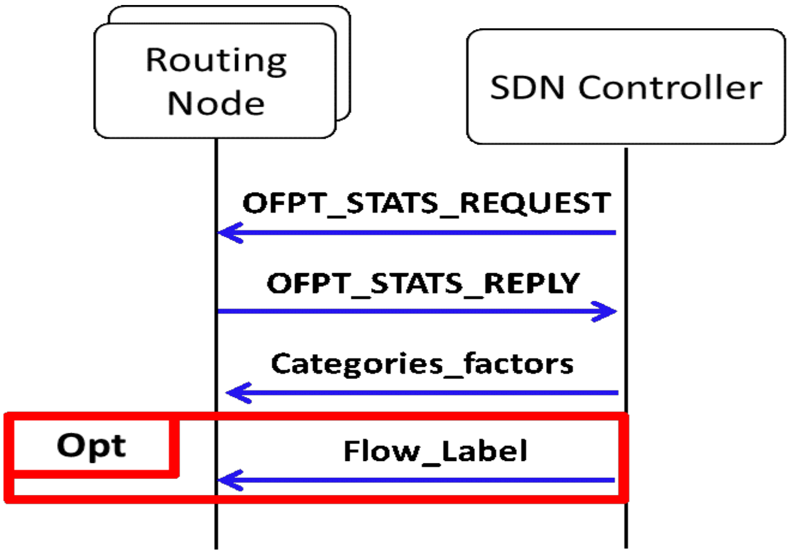


Figure 4.10: Message exchange when the AI module updates the different factors used in the metric.

4.2.5 Methodology and Results

In this subsection, the experiments done are described. First, the topology and the scenarios tested are shown. Then, the results are displayed.

Topology, Simulation Environment and Scenarios

The topology used in the experiments is shown in Fig. 4.12. Different scenarios provoke changes in the characteristics of the links, but all the scenarios can be implemented in the same topology. The links will change their characteristics according to the scenario, but the structure of the available paths will always be like the one depicted in Fig. 4.12.

In our topology, we have 6 switches, forming different subnetworks. Each subnetwork can be composed by several computers. However, in the simulation, each network is replaced by one single PC. The simulation has been done through Mininet. Mininet emulates both, the PCs and the switches, as Linux hosts. All the elements in the network are labeled and will be referenced in this text as they are in Fig. 4.12. Regarding the scenarios, five different situations have been chosen to test the proposal. In the first one, the network is used

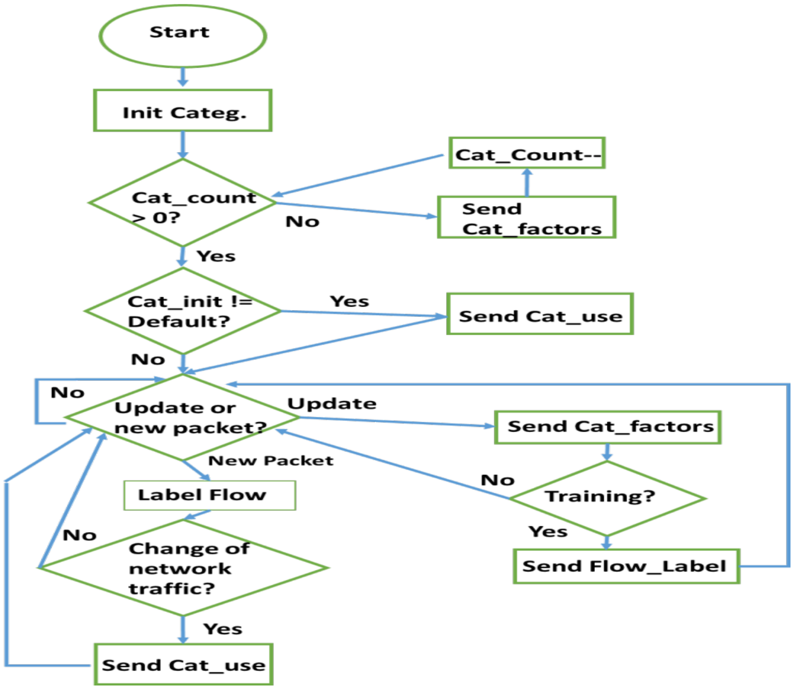


Figure 4.11: Algorithm diagram message exchange.

for multimedia streaming with enough resources to handle the transmissions. Then, the second scenario consists in handing several multimedia streaming flows that can consume too much network resources.

In the third scenario, the traffic in the network is not only composed by multimedia streaming flows, but also by VoIP traffic, being its main kind of traffic. The link with the greater bandwidth has also a great delay that can reduce the QoS of the VoIP traffic.

Scenarios 4 and 5 are designed to take into account the loss rate. In scenario 4, a TCP traffic of a file-downloading flow is sent in a network where the multimedia streaming flow is the main traffic. The link with the greatest bandwidth has enough loss rate to reduce the QoE of the multimedia traffic. The fifth scenario differs to the fourth in using the optional feature, along with the Flow-label message to manage in different ways the TCP traffic from the UDP multimedia streaming flows. Table. 4.1 summarizes the features of the different scenarios.

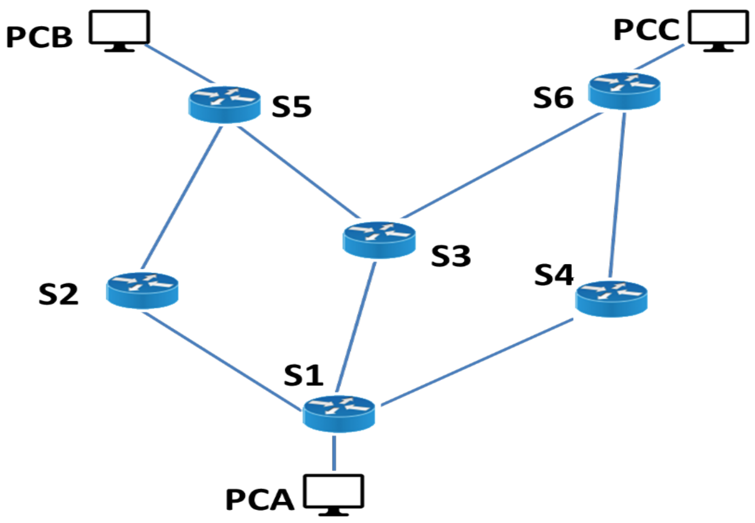


Figure 4.12: Topology used in the experiments.

In each scenario, the performance of the multimedia flow is measured in terms of QoS by using Wireshark at the destination side. In both cases, OSPF and the proposal, the performances are tested and compared.

The results of each scenario are discussed in the next subsection.

Results

In this subsection, the results obtained from each scenario are shown. Fig. 4.13 shows the bandwidth in bits per second obtained in scenario 1, where the multimedia flows do not exceed the maximum bandwidth. Both flows consume similar bandwidth, with maximums of 3.419 Mbps and 3.266 Mbps and minimums of 6.2 kbps and 6.4 kbps for the OSPF test and for the proposal respectively. The average of the OSPF transmission is 1.070 Mbps and the average of the proposal is 1.015 Mbps.

In Fig. 4.14, the delay of each packet in ms with OSPF and the one produced with the proposal are shown. Both delays are quite similar, having maximums of 19.74ms and 19.85ms respectively and minimums of 0.04ms and 0.03ms. The average is also similar, 6.521ms in the OSPF transmission and 6.541ms by using the proposal.

Table 4.1: Scenarios used for testing.

	Scenario 1	**Scenario 2**	**Scenario 3**	**Scenario 4**	**Scenario 5**
Flow sent	Multi-media	Multimedia	Multimedia, VoIP	TCP, multimedia	TCP, multimedia
Network resources	Enough	Not enough	Enough bandwidth but high delay	Enough bandwidth but with losses	Enough bandwidth but with losses
Flow-label optional feature	No	No	No	No	Yes

In terms of jitter, Fig. 4.15 shows that they are very similar too. The jitter of OSPF reaches 1.2ms and the jitter of the proposal reaches the 0.86ms. Their minimums are 0.01ms for both transmissions. The average jitter is 0.065ms for the OSPF transmission and 0.66ms for the proposal.

Regarding scenario 2, where the multimedia flows exceed the capabilities of the link, Fig. 4.16 displays the delay and Fig. 4.17 the jitter. We have observed that the delay is similar. The paths, where the traffic is sent through, present similar latencies. In both transmissions, the maximum delay is 19.75ms. However, the minimum delay obtained is different. The OSPF transmission presents a minimum delay of 1.8ms and the proposal obtains 0.02ms of minimum delay. The average delay with OSPF is greater than the one obtained in the transmission using the proposal. OSPF transmission shows a 6.984ms of average delay, while the proposal achieves 6.5ms.

As it is shown in Fig. 4.17, the jitter produced during the transmission is higher in the OSPF case, presenting fluctuations and increasing up to 9.48ms of maximum, while the proposal presents a maximum of 1.63ms, reducing the maximum jitter in 83%. The minimum jitter is 0.01ms in both transmissions, but the greatest difference is presented in the average jitter. The average jitter of the OSPF transmission is 1.396ms while the average of the proposal is 0.082ms. There is a jitter reduction of 93%.

Finally, Fig. 4.18 shows the bandwidth consumed in each case. With the path selected by OSPF, the bandwidth is not able to be grater that 1.961

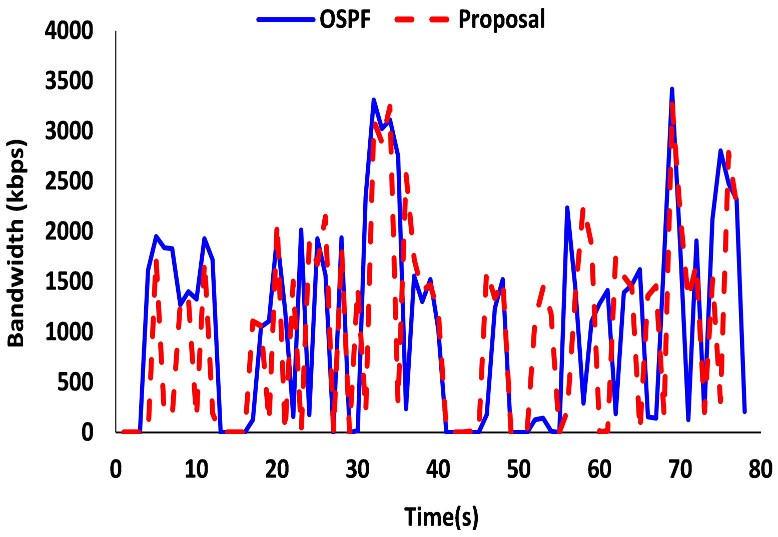


Figure 4.13: Bandwidth in bits per second registered in scenario 1.

Mbps, which is the available bandwidth during the transmission. However, by selecting the path according to the available bandwidth, the proposal reaches 3.244 Mbps, avoiding QoE problems and increasing the throughput in 40%. The minimum throughput in the OSPF transmission is 6.4 kbps and with the proposal is 6 kbps. Moreover, the average bandwidth is 1.181 Mbps for OSPF and 1.2 Mbps for the proposal.

In scenario 3, there is a difference in terms of delay between the path of the highest bandwidth and the one chosen by the proposal for VoIP transmission. Fig. 4.19 presents its results as regards delay. OSPF transmission average delay is 558.33ms, while the proposal transmission suffers an average delay of 58.29ms. The maximum delay of the OSPF transmission is 612.55ms and the maximum for our proposal is 108.52ms. The minimum delays are 500ms for OSPF and 0.09ms for the proposal.

The jitter results are presented in Fig. 4.20. They do not differ too much. Both transmissions have a minimum jitter of 0.01ms. The average jitter is also similar with 0.0412ms for the OSPF transmission and 0.0427ms for the proposal. Maximum values are 1.13ms for the OSPF and 0.55ms for the proposal.

In scenario 4, the path with the greatest bandwidth also presents a high loss rate. Fig. 4.21 shows the delay obtained from both transmissions. OSPF

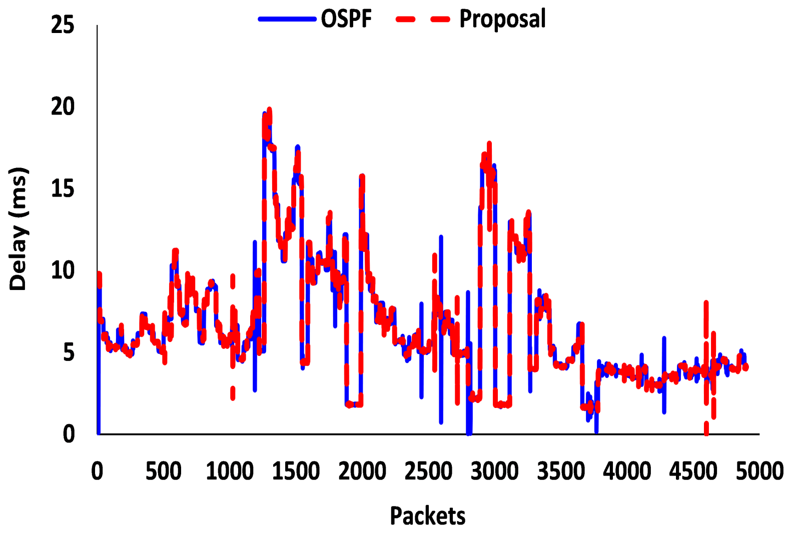


Figure 4.14: Delay in ms registered in scenario 1.

transmission presents a greater average delay of 8.05ms, while the proposal presents a reduction, obtaining 7.08ms of average delay. The maximum delay is also increased to 59.58ms with OSPF, having a maximum of 52.5ms with the proposal. Nevertheless, the minimum delay is 1.25ms in the OSPF case and 1.83ms with the proposal routing solution.

Fig. 4.22 shows the jitter. There is a minimum of 0.01ms in both cases. The differences are presented in maximum and average jitter. OSPF presents an average of 1.51ms, while the proposal achieves 1.4ms. The maximum jitter presented in the transmission using OSPF as a routing protocol is 10.92ms, more than 1ms greater than the maximum with the proposal, which presents 9.67ms.

Finally, Fig. 4.23 displays the loss rate of each transmission. The characteristics of the links and the routing decision of OSPF through the maximum bandwidth path cause an increment of loss rate in OSPF transmission. 15% of the multimedia packets are lost during the transmission. With the proposal, the packets are sent through a path with less loss rate, achieving 3% of loss rate.

Scenario 5 shows the capability of making different decisions depending on which kind of traffic is being sent. As it is explained in Subsection 4.2.4, this

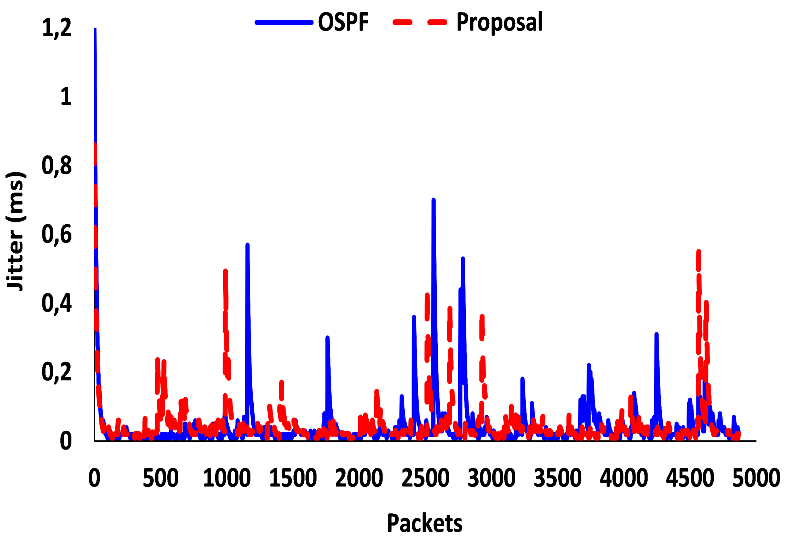


Figure 4.15: Jitter in ms for each packet in scenario 1.

functionality is not the central point of the proposal, but it can be applied. This fifth scenario tests the load balancing done when the metric is changed in every kind of flow. Fig. 4.24 shows that this difference in the metric calculation allows the streaming to exceed the 2 Mbps, having a maximum of 3.27 Mbps of throughput. Without taking individual decisions, the maximum throughput is 1.95 Mbps. Minimum throughput is also increased. Without this characteristic, the minimum throughput is 0.44 kbps and when it is used, it raises to 6.58 kbps. Average bandwidth also increases from 1.17 Mbps to 1.22 Mbps.

In addition, Fig. 4.25 shows a jitter reduction due to the possibility of sending at the maximum bitrate, without limitations. The average jitter is reduced from 1.4ms to 0.1ms, more than 90% of reduction. The maximum jitter is also reduced from 9.67ms to 1.31ms, presenting a reduction of 86%. The minimum jitter is 0.01 in both scenarios.

Table. 4.2 summarizes the results obtained in the tests. For each scenario, the different values of jitter, bandwidth, delay and loss rate are detailed. The first value is for OSPF and the second one for our proposal in all the scenarios except in the last one. In that scenario, the first value is the one obtained without using the optional feature and the second one is the gathered when that feature is activated.

Table 4.2: Results of each scenario using OSPF and our proposal.

	Scenario 1	**Scenario 2**	**Scenario 3**	**Scenario 4**	**Scenario 5**
Minimum Jitter	0.01 / 0.01ms	0.01 / 0.01ms	0.01 / 0.01ms	0.01 / 0.01ms	0.01 / 0.01ms
Average Jitter	0.065 / 0.066ms	**1.396 / 0.082ms**	0.0412 / 0.0427ms	1.51 / 1.4ms	**1.4 / 0.10ms**
Maximum Jitter	1.2 / 0.86ms	**9.48 / 1.63ms**	1.13 / 0.55ms	10.92 / 9.67ms	**9.67 / 1.31ms**
Minimum bandwidth	6.2/6.4kbps	6.4/6kbps	-	-	**0.44 / 6.58kbps**
Average bandwidth	1.070 / 1.015Mbps	1.181 / 1.2Mbps	-	-	1.17 / 1.22Mbps
Maximum bandwidth	3.419 / 3.266Mbps	**1.961 / 3.244Mbps**	-	-	**1.95 / 3.27Mbps**
Minimum delay	0.04/0.03ms	**1.8/0.02ms**	**500/0.09ms**	1.25/1.83ms	-
Average delay	6.521 / 6.541ms	6.984/6.5ms	**558.33 / 58.92ms**	8.05/7.08ms	-
Maximum delay	19.74 / 19.85ms	19.75/19.75ms	**612.55 / 108.52ms**	59.58/52.5ms	-
Loss rate	-	-	-	15/3(%)	-

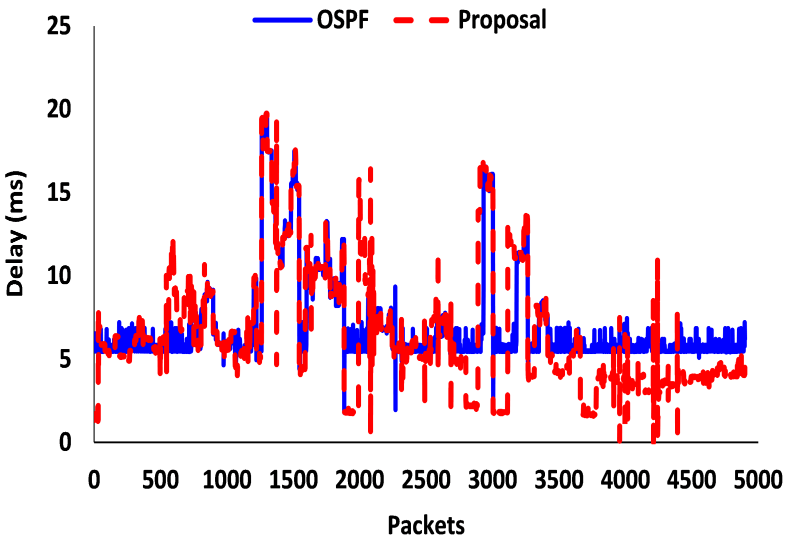


Figure 4.16: Delay in ms obtained in scenario 2 for each packet

4.2.6 Comparison with other OSPF variations

In this subsection, the proposal is compared with other OSPF variations. This comparison is displayed in Table. 4.3. On the one hand, in [166], Ye et al. proposed a metric based on some queueing models focused on packet loss. They improved the packet loss rate in 30-60% depending on the topology. Their proposal reduces the loss rate and tries to get the best value of the metric factor within a limited time frame, but they cannot take into account several factors and adapt their weights depending on the use of the network. We can do the same but achieving a completely adaptive routing solution.

On the other hand, O'Halloran and Chambers use the network load in order to provide an adaptation of OSPF interface metrics in [167]. They try to dynamically modify the OSPF interface cost of the routers. However, technical problems impede them to achieve the expected performance. With the use of SDN, we can solve their problems, being aware of the number of flows and the destination of each flow.

Table 4.3: Comparison of OSPF modifications.

	Performance Improvement	**Adaptative Metric**	**AI Metric Definition**	**Technology Used**	**Multiple Factor Metric**
Ye [166]	No Data	External Updates	No	External program with technical problems	No
Chambers [167]	30-60% loss rates	Yes	No	Theoretical queuing model	No
Our Proposal	80% loss rate Up to 90% Jitter Reduction Increment of 40% of Throughput	Yes	Yes	SDN	Yes

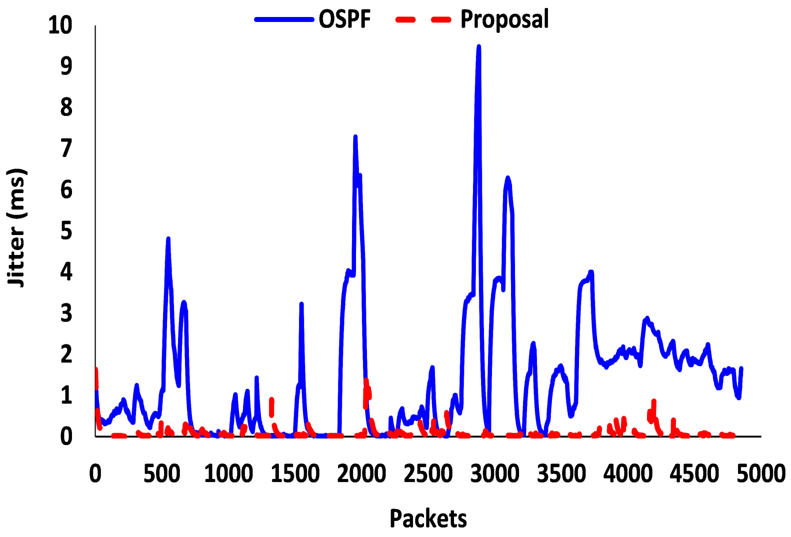


Figure 4.17: Jitter in ms for each packet in scenario 2.

4.2.7 Conclusion

Routing decisions has been traditionally taken depending on factors related to capabilities of the paths or the distance between source and destination. However, these solutions did not take into account the actual use of the network. Each kind of network in each interval of time can need some different resources. Moreover, the importance of these resources also changes through the use of the network. SDN adds adaptability to the network and can be used to propose new kinds of applications and routing solutions. In this paper, we have proposed a dynamical metric solution in order to provide the network the ability to choose the path depending on the main use of the network.

Results show that, when the use of the network does not need more resources in terms of available bandwidth, OSPF and the proposal achieves similar results (Fig. 4.13 and Fig. 4.14 and Fig. 4.15). However, being able to adapt the metric factors values depending on the use of the network can make use of higher bandwidth when several multimedia streaming flows are present in the network. In the experiments, OSPF chooses a path when the transmission can only use up to 2 Mbps. Nevertheless, our proposal uses load balancing that allows the multimedia streaming to use all the required bandwidth, with an increment of 40% of the throughput. In addition, there is also a jitter reduction of 93%. The third scenario demonstrates that changing the importance of

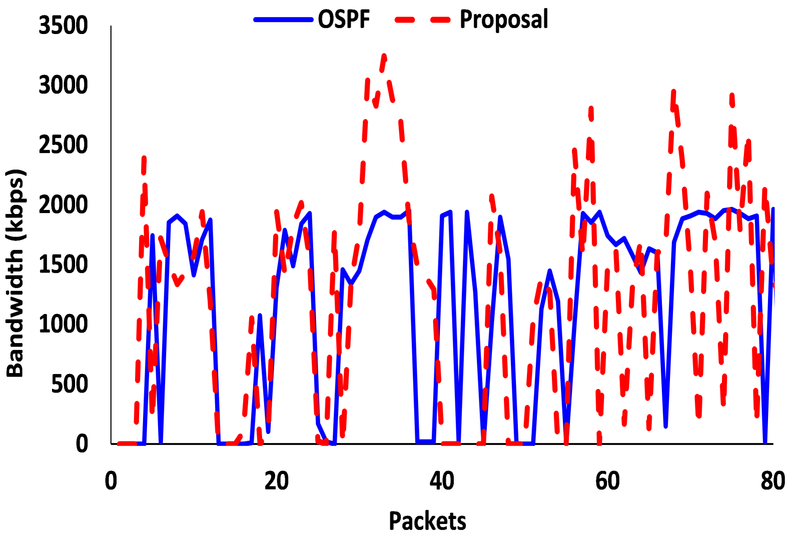


Figure 4.18: Bandwidth in bits per second in scenario 2.

every factor according to the main kind of traffic can improve the quality of the transmission. The proposal changes its metric to prioritize delay reduction. Fig. 4.19 shows a reduction of more than 400ms. Finally, loss rate can be also taken into account. Moreover, in TCP and multimedia streaming flows, using a different metric for each kind of traffic has achieved a 90% reduction of the average Jitter in scenario 5. In addition, it also allows the possibility of using more than 3Mb/s of bandwidth in the multimedia streaming, using a path with an 80% less of loss rate.

The comparison with the other related works shows that our proposal is the only one that introduces AI and SDN to modify OSPF protocol. With these modifications, we achieve real time improvements and adaptive routing, which is currently needed to provide real-time services (see [168] and [169]).

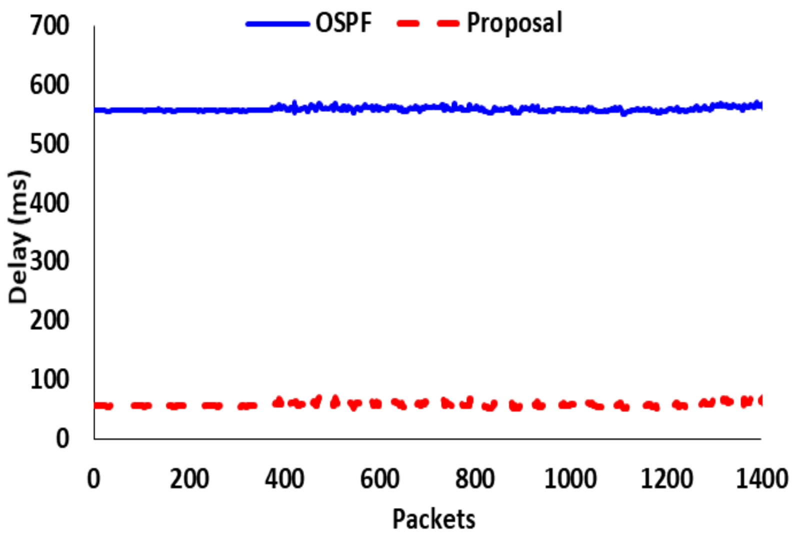


Figure 4.19: Delay in ms obtained in scenario 3 for each packet.

4.3 Intelligent SDN Routing Protocol

The proposal of modifying the OSPF routing protocol has introduced some advantages that improves the QoS of multimedia services. However, a specific metric based on the well-known multimedia QoS parameters might not achieve an optimal transmission in some scenarios. The limitations of OSPF, especially the variety of actions it may take to overcome the problems during transmission, may affect the quality of the transmission.

In this section, a new routing proposal that takes advantage of all the capabilities of SDN is described. First, a brief description of the aim of the routing protocol is provided. Then, the definition of the different data structures that the protocol must manage is described. Next, the diagram state and the main algorithm are detailed. After that, the messages needed to implement the protocol are listed. Finally, the section is concluded with a brief conclusion.

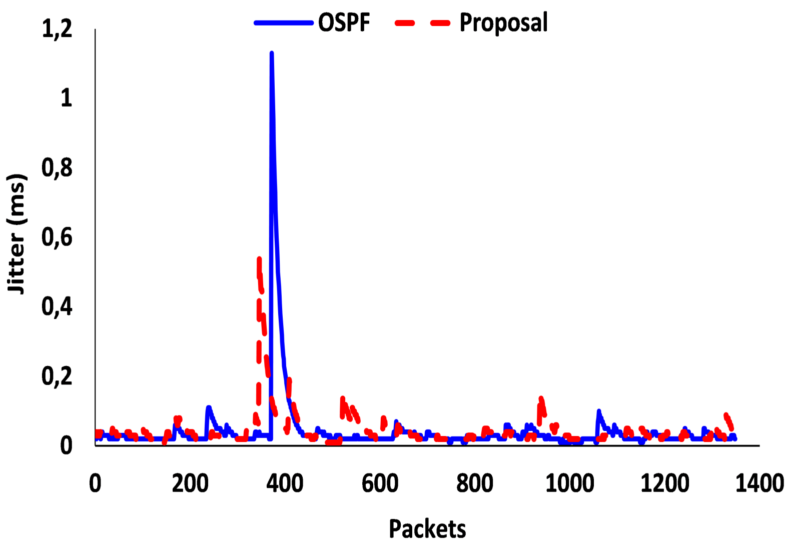


Figure 4.20: Jitter in ms for each packet in scenario 3.

4.3.1 Aim and scope

The main goal of the routing protocol proposed in this section is to provide the best QoS of the multimedia SDN flows in the most various number of scenarios. In order to achieve that, the capabilities of the SDN controller to perform actions and changes in the network are a key factor. The base of the routing protocol will be SDN. The architecture underlying the protocol is the one explained in the introduction and depicted in Fig. 4.1.

The routing module can be either an internal module of the SDN controller, or a northbound application. In this chapter, both possibilities are addressed. The messages used by the routing module and its interaction with the SDN controller may vary depending on whether it is a core module or an external application. Nonetheless, in this chapter both interactions and messages will be detailed.

The routing module is strongly based on the AI module shown in Fig. 4.1. However, its implementation will be discussed separately in chapter 5. This allows the routing module to be independent of the intelligent method used by the AI module. If new methods or more detailed studies are performed or discovered, they can replace the one chosen in this work, and the routing

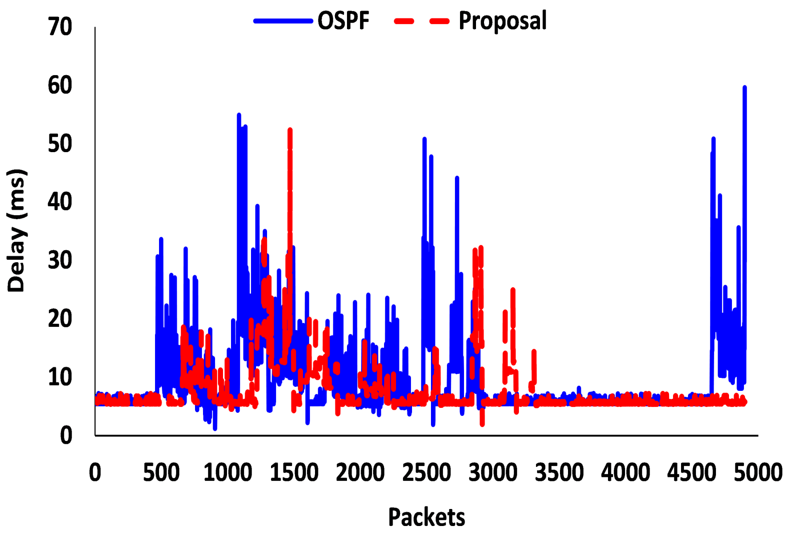


Figure 4.21: Delay in ms obtained in scenario 4 for each packet.

module can still work. Due to that, the performance test of the proposal will be located at the end of chapter 5.

4.3.2 Data Structures

In this subsection, the data structures strictly used by the routing algorithm are discussed. The ones used by the intelligent system proposed are described in chapter 5.

The first structure needed by the routing algorithm is the topology. This means, the routing algorithm must be aware of the network and the nodes and links that are part of it. For that reason, the routing module has to manage an array of flows. These flows are references of the flows managed by the SDN controller. Therefore, the content of the array is an integer that identifies the flow. If the routing module is a core module, this identifier is directly obtained from the objects of the controller. If it is an external application, the SDN controller must provide the identifiers of the flows of a specific node, or all the flows of the whole network, through a request.

In addition to the flows, another array of multimedia flows is needed. This one is used to take into account the flows that are really important to the

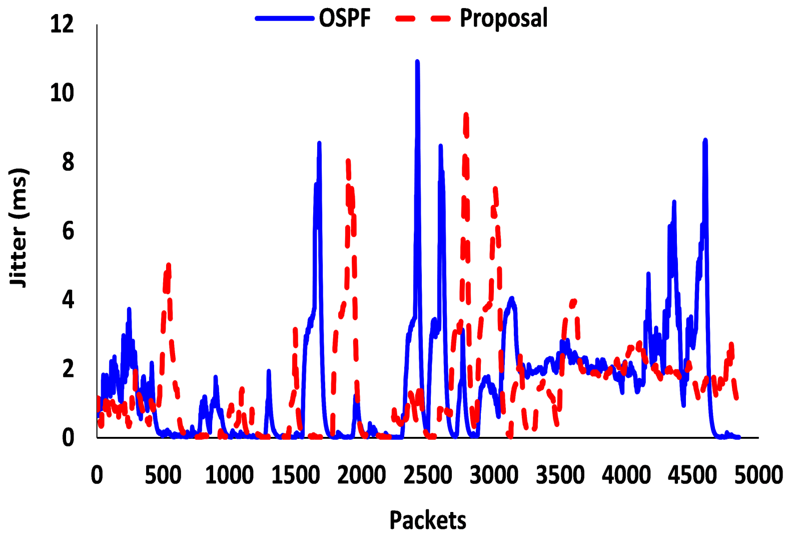


Figure 4.22: Jitter in ms for each packet in scenario 4.

algorithm, those flows that determines the QoS. Hence, the routing algorithm must improve the transmission of those flows.

Additionally, the routing algorithm needs a way to manage routes in the network. An array of possible routes is added to the routing algorithm. This is built from the topology and it contains an array of links, a cost in terms of initial metric and the state of the route, important to know if any multimedia flow can be set to the specified route or not. This state is not limited to up and down states, but also to congestion states when it may be beneficious to select one of the flows in the route and select a new route. For that, the routing algorithm also needs a list of flows that goes through each route.

Table. 4.4 summarizes the main data structures needed by the routing module. There, a name, the type and a brief description are provided.

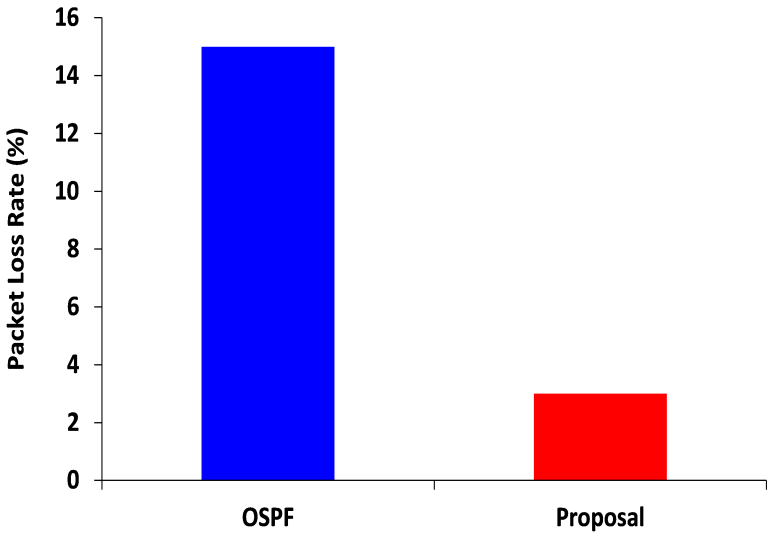


Figure 4.23: Loss rate presented with OSPF and the proposal in scenario 4.

4.3.3 States and algorithm

In this subsection, the main algorithm of the routing protocol, along with the states diagram, will be explained. For that, first, the task and the behavior of the different modules must be discussed.

First, it is important to highlight that the routing protocol is designed to solve problems with the multimedia transmissions. However, at the beginning of the execution, the routes from a source to a destination must be set. Any algorithm, like Dijkstra can be used for that goal. Nevertheless, the proposal takes advantage of the SDN architecture for the very beginning to get the better performance of the multimedia flows. Using the global point of view of the SDN controller allows the routing module to avoid the exchange of specific control messages as the ones in OSPF or other traditional routing protocols. When the system starts, the SDN controller builds the topology and the necessary flow entries to provide the communications between the nodes. Then, when a new multimedia flow is detected, a route is assigned based on the classical QoS parameters, i.e., bandwidth, delay and jitter. The cost of the routes is calculated using (4.5).

Until this point, the working of the routing algorithm seems quite similar to the OSPF modification described in the previous section. Nonetheless,

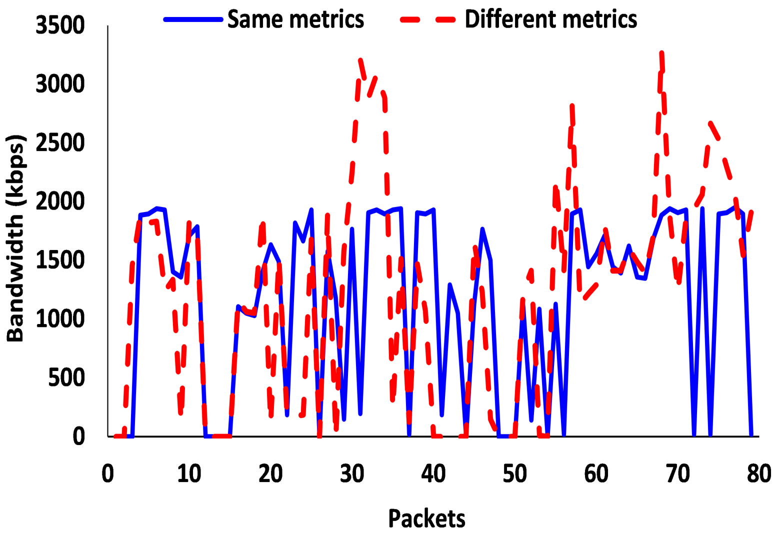


Figure 4.24: Bandwidth consumed for each transmission, by using the same metrics for each flow and by using different metric for the different flows.

the dynamic of the route selection is quite different. Once the multimedia transmissions suffer a problem that may reduce their QoS, and, hence, their QoE, the routing algorithm modifies the behavior of the network to solve it, even if the links still work. This is achieved thanks to the AI module. Unlike the OSPF modification, where the AI module only set appropriate values of the coefficients of the route cost calculation, in this proposal it plays the main role. The AI module is the application that chooses how to act given a specific problem in the network. To sum up this first explanation, the tasks of each module are depicted in in Fig. 4.26. It can be seen as the routing module is the one that manages the route information, or the topology information, and the statistics. Furthermore, it is the routing module which performs the changes in the network or in the multimedia transmissions. On the other hand, the AI module is the module that is in charge of taking decisions based on data and intelligent algorithms.

Fig. 4.27 depicts the different parts of the routing protocol, as in Fig. 4.2 and Fig. 4.3. The first task is the routing table creation. Since the routing protocol works over an SDN, this means that the flow table is filled with flow entries decided by the Routing Module. It will be detailed when the states are defined later, but, as it was said before, in the beginning, that means to fill the flow

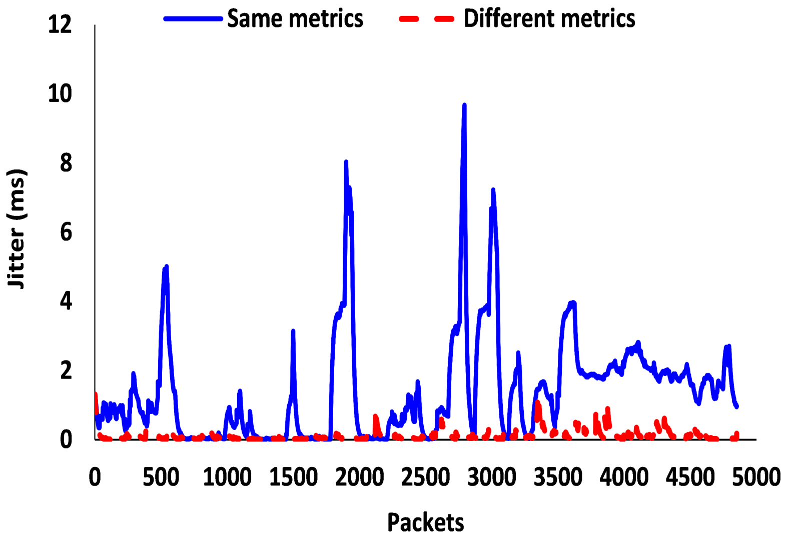


Figure 4.25: Jitter obtained in scenario 5.

tables with entries that allow sending data from the source to the destination with the maximum QoS. Moreover, in order to do that, the routing module uses a global view of the network topology. It is important to differentiate here between the routing module as a core service and as an external application. The topology management is one of the key features of an SDN controller. Consequently, if the routing module is a core service of the SDN controller, it can access to the topology information. On the other hand, it is an external application, the API of the SDN controller must provide the application with the possibility of getting the topology data (e.g., with a GET call in a REST API).

The next functionality implemented by the routing module is the statistic collection. This is utterly important to provide the AI module with an accurate current state of the network. As it was described in chapter 3, the statistics gathering is placed in the SDN controller. Therefore, if the routing module is a core service, it can easily access to the statistics gathered by the SDN controller. If it is a northbound application, the API must provide at least three different methods to get the needed data to manage the state of the network. In a REST API they would be three GET URIs, one for the flow statistics, one for the statistics of a link and one for the statistics of a specific node. For all these functions, the routing module needs to provide the SDN

Table 4.4: Data structures used by the routing module.

Name	Type	Description
Topology	Array Switch Array Links	An object that represents the topology of the network
Flows	Array FlowMod	An array with all the flows of the network
Multimedia Flows	Array FlowMod	An array with all the multimedia flows of the network
Routes	Array Links Integer Flow Integer State	Array of routes (array of links) associated with the flows that go through it
Possible Routes	Array Links Switch Source Switch Destination QoSMetrics metrics Integer State	Array of possible routes that a flow can take from a source to a destination the current state and the performance of that route

controller with an identifier of the object which he wants to get the stats. Additionally, the SDN controller can implement another function to send the stats of every object in the network.

Another important functionality of the routing module is the metric calculation. The QoS values of the flows must be calculated to assign the first route in the beginning of the execution. However, they are also important to the AI module, which may need them to evaluate if the multimedia transmissions suffer from QoS degradation and they need some actions to be performed. Hence, the routing module sends this QoS metrics to the AI module. As a result, the AI module returns a set of actions that will be implemented by the routing module, or that will be sent to the SDN controller if it is an external application. This represents the last functionality depicted in Fig. 4.27. The routing module defines the set of actions that will be available for the AI module to choose. These actions may be available or not depending on the topology. That is another reason why the routing module needs to have access to the topology. For instance, if the routing module does not know that it exists two different routes from the source node to the destination, changing to an alternative route could not be set as a possible action to be performed. Moreover, it must translate the actions to a set of flow modifications. Therefore, the data sent to the SDN controller, in case the routing module is a northbound

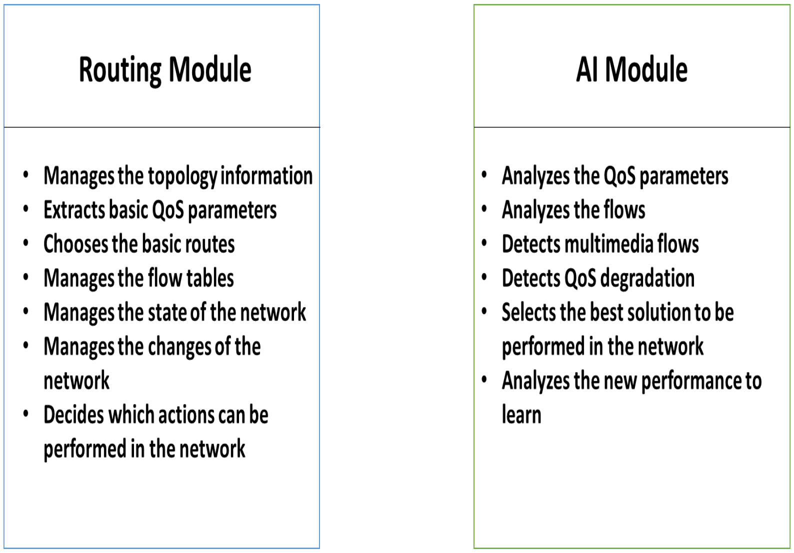

Routing Module
• Manages the topology information
• Extracts basic QoS parameters
• Chooses the basic routes
• Manages the flow tables
• Manages the state of the network
• Manages the changes of the network
• Decides which actions can be performed in the network
AI Module
• Analyzes the QoS parameters
• Analyzes the flows
• Detects multimedia flows
• Detects QoS degradation
• Selects the best solution to be performed in the network
• Analyzes the new performance to learn

Figure 4.26: Module tasks.

application, are a set of OpenFlow messages. They are, specifically, flow mod messages. For example, if the action chosen is to change to an alternative route, the routing module selects, from the possible routes array, the next route with best metrics. Then, it modifies the flow entries to forward the flow packets through the new route. All the messages used by the routing protocol to perform the actions or to communicate with the AI module are explained in the next subsection. However, Fig. 4.28 shows the communication exchange between both modules, which summarizes the explanation given so far.

At this point, the way of working of the routing algorithm can be depicted in a state diagram (Fig. 4.29). In the beginning, the routing module has no data about the topology and it must build up or it must receive it from the SDN controller ("No Info" state). In order to do that, the routing module creates flow entries to get information about the topology. When the topology data is ready, the routing algorithm calculates the first routes and uses the SDN controller to fill the flow tables on the nodes and initializes the AI module. Now, in this state, no multimedia flows are being transmitted in the network. While this condition is true, the route previously calculated will be used and the new flows are sent to the AI module. When the first multimedia flow is detected, the statistics will start being transmitted to the AI module.

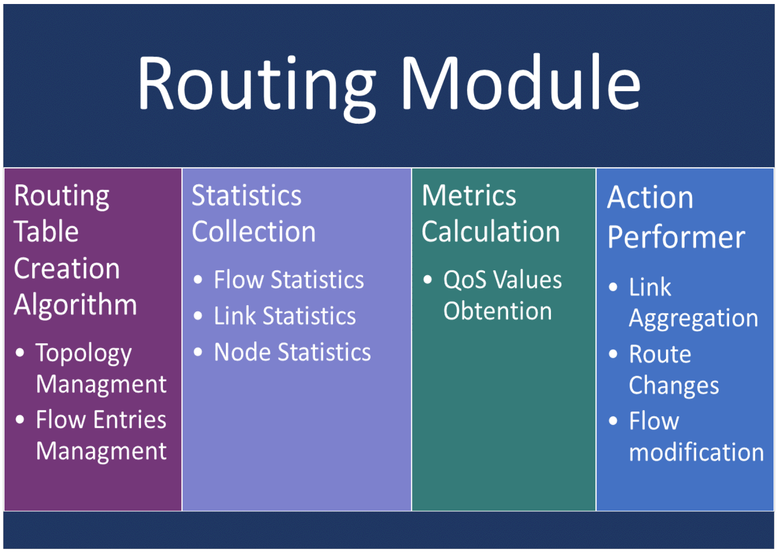


Figure 4.27: Routing Module submodules.

Once there are multimedia flows in the network, the AI module may notify the routing module with a warning of QoS degradation, which will indicate the set of actions to perform. The routing module will perform those actions and will notify the AI module that the actions have been performed. Then, it will enter into the state "Checking Upgrade". This state lasts for a determined delta period of time and, during this time, the routing module will provide the AI with periodic QoS metrics. These measurements may be used by the AI module to improve the decision of which action should be performed in the future. This will depend on the intelligent method used by the AI module. In the method proposed, the AI module learns from the past decisions.

Algorithm 4.2 describes in pseudocode the working of the routing module. Since both possibilities, the module being a core service and being a northbound application, are being taken into account, in the algorithm, some lines are specific for one of the options and it is explained. As regards the functions used in the code, some explanations can help the reader to understand how are implemented:

- The send and notification functions simply listen or uses sockets to the AI application or the controller. If the module is a core service, just writes or read a shared variable of the specific class.

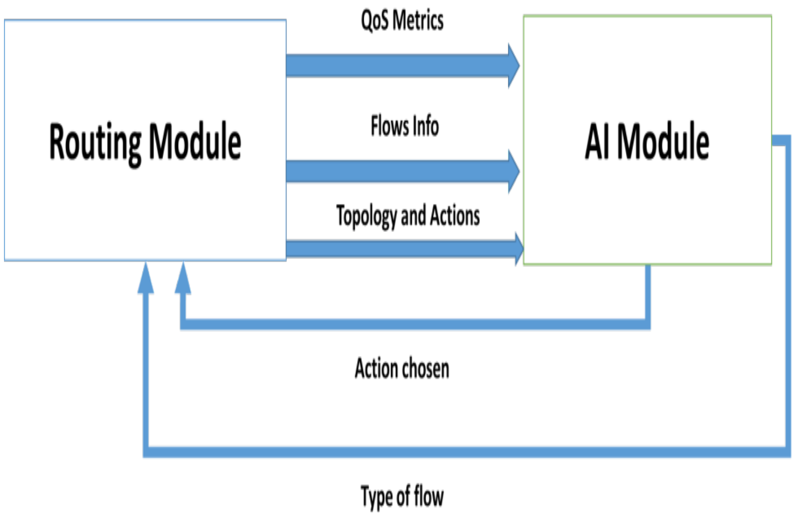


Figure 4.28: Data exchange between modules.

- Start_QoS_Measurements() creates a new thread that, periodically, get statistics from the nodes and calculates the QoS parameters.
- Set_Route adds the flow entries needed to direct the flow through the desired route based on the QoS parameters and (4.5). It also adds the flow to the routes array.
- Perform_Action sends the OpenFlow messages (or request that sending to the controller if the module is an external application) needed to perform the action notified by the AI module. The actions are listed in chapter 5, where the AI module is described.

4.3.4 Messages

In this subsection, the messages needed to implement the behavior described in the previous subsection are described. It is important to take into mind that some of the messages will be only be useful if the modules are external applications. Since in this chapter that possibility is being studied, those messages are also described in this subsection.

The first messages that are going to be explained in this subsection are the ones needed if the routing module is an external application and, therefore,

Algorithm 4.2 Routing Module Algorithm

```
Given: Topology, PossibleRoutes // If it is a core service
Request_Topology(); // If it is an application
Set_Initial_Routes() ;
Send_Possible_Actions_AI();
deltaTime = 0;
While true
    If deltaTime > 0
        deltaTime -= DELTAINCREMENT;
        If deltaTime <= 0
            Send_QoS_AI();
        End If
    End If
    If Get_New_Flow_Notification()
        Add_Flow();
        Multimedia = Send_Flow_To_AI();
        If Multimedia is true
            Add_Multimedia_Flow();
            If MultimediaFlows.size == 1
                Start_QoS_Measurements();
            End If
        End If
        Set_Route();
    End If
    If Get_AI_Notification() == QOS_DEGRADATION
        Perform_Action(Notification.Action)
        deltaTime = DELTA;
    End If
    If Get_Expired_Flow_Notification()
        RemoveFlows();
        If MultimediaFlows.size == 0
            Stop_QoS_Measurements();
        End If
    End If
    If Get_Topology_Notification()
        Update_Topology(); // If it is a core service
        Update_PossibleRoutes(); // If it is a core service
        Request_Topology(); // If it is an application
        Calculate_PossibleRoutes();
        Send_Possible_Actions_AI();
    End If
End While
```

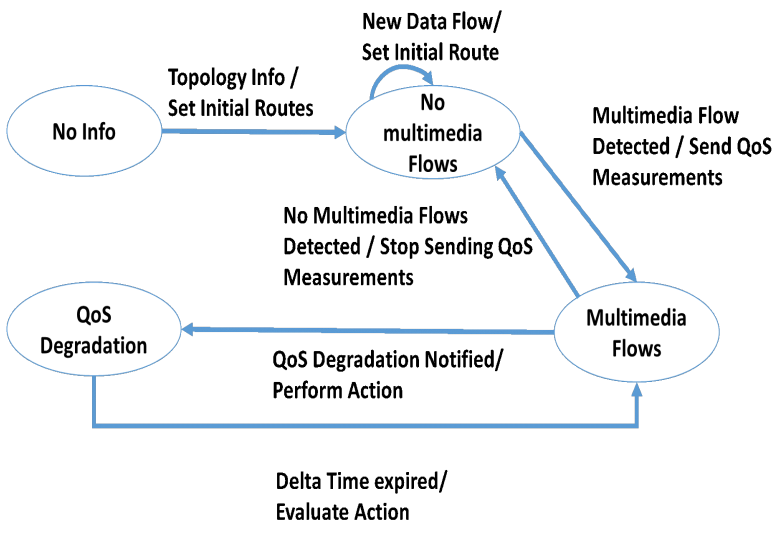

New Data Flow/
Set Initial Route
Topology Info /
Set Initial Routes
No Info
No multimedia Flows
Multimedia Flow Detected / Send QoS Measurements
No Multimedia Flows Detected / Stop Sending QoS Measurements
QoS Degradation
Multimedia Flows
QoS Degradation Notified/ Perform Action
Delta Time expired/ Evaluate Action

Figure 4.29: States Diagram.

it cannot directly access to the SDN controller data. Furthermore, it is going to be supposed that the API is a REST API. Consequently, the messages are HTTP messages and the objects are described in JSON.

- Request Topology: This message is a GET message from the routing module to the SDN controller. It is used to request the whole topology of the network. It does not need further content. The routing topology uses this message to get the topology from the SDN controller at the beginning of the execution.

- Send Topology: As a response of a Request Topology, the SDN controller sends to the routing module the whole topology of the network. This is a POST message, which body contains all the nodes and the links data in a JSON. The routing module builds the topology with this data and calculates the possible routes from a source to a destination with the Dijkstra algorithm.

- Request Flow Notifications: The routing module sends this message as a request to the SDN controller to get notifications about new and expired flows in the network. This is a GET request that the routing module uses to get subscribed to the SDN controller flow notifications.

Table 4.5: Values for type and len of each message.

Messages	Type	Len
RTG_SET_ACTIONS	0	6
RTG_QOS_PARAMETERS	1	18
RTG_FLOW	2	6 plus the length of the Flow Mod structure
RTG_PERFORM_ACTION	3	2
RTG_MULTIMEDIA	4	6
RESERVERD	5	-
RTG_NEW_ACTION	6	2

- Notify New Flow: This message is a notification about new flows in the network. The SDN controller sends a POST message to the routing module with a JSON, which contains the data of the new flow. The routing module adds the flow to the flows array and send it to the AI module to determine whether it is a multimedia flow.
- Notify Flow Expiration: This message is quite similar to the previous one. It is a flow notification from the SDN controller, and it is a POST message that contains the flow id as a body. However, this message is used by the routing module to remove the flow from the array and, if it is the last multimedia flow in the network, to stop the QoS statistics gathering.
- Send FlowMod: The last message of this list is a POST message sent from the routing module to the SDN controller. It is used by the routing module to provide the SDN controller with a set of OpenFlow messages, flow mod messages, to send to the different nodes in the network. The body of the POST message is a JSON where the OpenFlow message and the destination of that node are specified. This structure is similar to the one shown in Fig. 3.18.

All the remaining messages are composed by the type of message, the length of the message and, then, the different fields needed to understand the message. The type field is a byte whose value determines which message is being sent. In this work, only six different messages are used. However, the type field can use higher values for future extensions of the protocol. The values for each messages are indicated in Table. 4.5.

Fig. 4.30 shows the first message of this type. It is the possible actions message or “RTG_SET_ACTIONS”. Its type value is 0. The set of possible actions is represented as a flag-based field of 32 bits length. Each bit represents an action, and its value signifies if the action can be performed in the current network or not. As it is indicated in the previous section, this message can be sent when the topology changes and one or several actions, then then, can be performed or cannot.

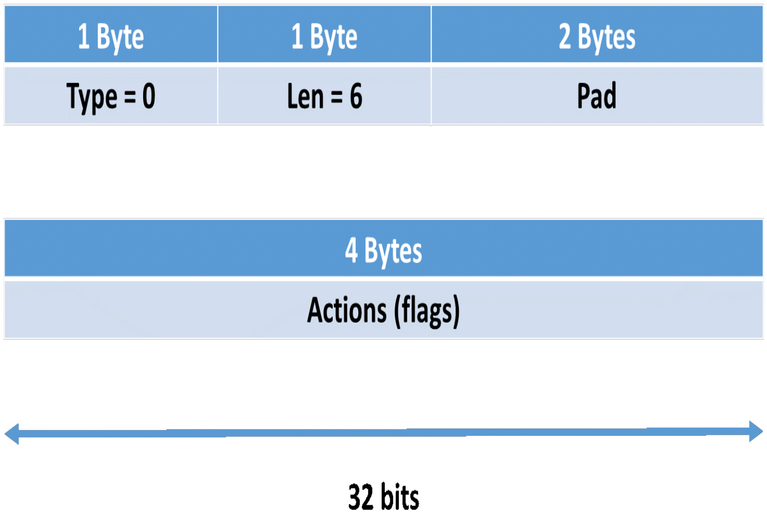


Figure 4.30: RTG_SET_ACTIONS message.

However, this message only works if the AI module manages the same actions that the routing module. In this work that is the case. Nevertheless, if the AI module it is replaced, this may not happen. For that reason, an optional message is included to allow specifying an action that the AI module has to manage. Fig. 4.31 depicts this message, called “RTG_NEW_ACTION”. Each action must define a set of arguments that are sent back to the routing module. For instance, the action of changing to a new route must provide the arguments of which flows must be changed.

The next message, shown in Fig. 4.32, is the “RTG_QOS_PARAMETERS”. This message is used by the routing module to send the measurements of the QoS metrics to the AI controller. The QoS metrics sent to the AI controller are the ones used in (4.5), i.e., bandwidth, jitter, delay and losses. Moreover,

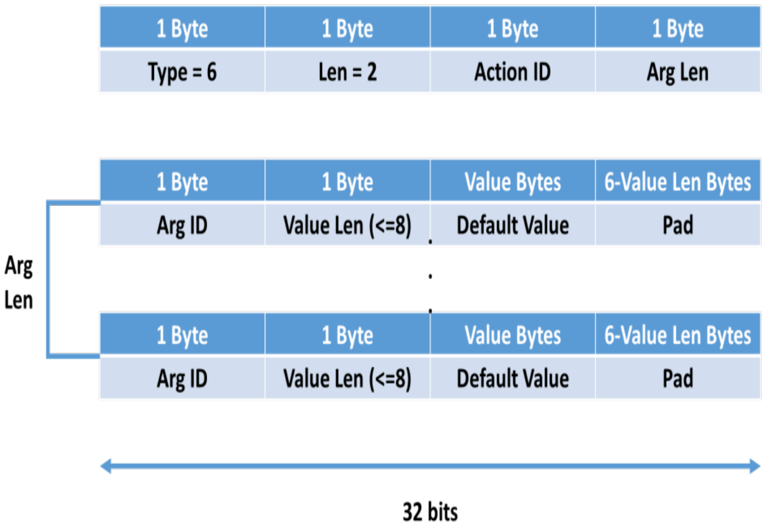


Figure 4.31: RTG_NEW_ACTION message.

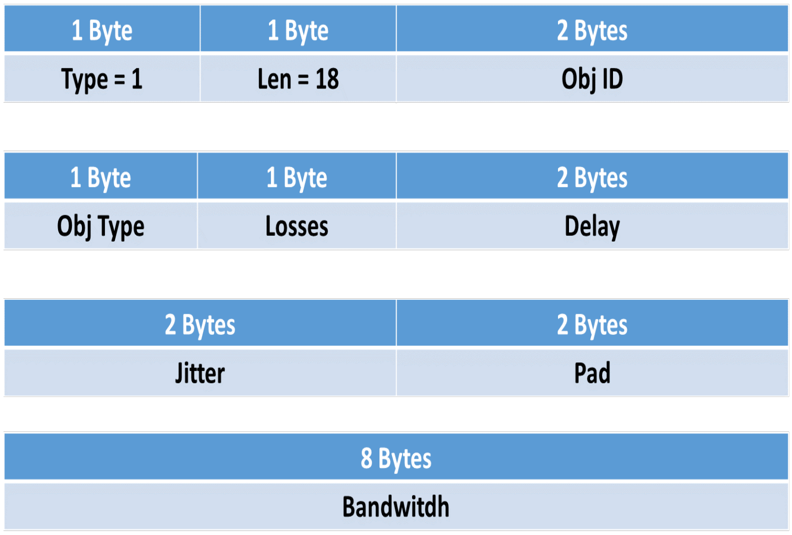
the type and the ID of the object with those metrics are also provided. That means, if the metrics belong to a node, to a route or to a flow and which one.

Fig. 4.33 shows the message used to provide the flows to the AI module. This is the "RTG_FLOW" message, which contains the ID of the flow, the ID of the node where it is allocated and the OF_FLOW_MOD that represents the flow.

The AI module notifies the routing module when a flow is detected as a multimedia. In that case, a "RTG_MULTIMEDIA" message is used. This simple message, depicted inFig. 4.34, contains the flow ID of the multimedia flow.

Finally, the structure of the message used by the AI module to indicate which action must be performed is described in Fig. 4.35. This message, called "RTG_PERFORM_ACTION" has the same structure than the optional "RTG_NEW_ACTION".

These are all the messages needed for the implementation of the routing module. However, some other communications could be implemented depending on the set of actions that are defined by the system. For instance, as it is described in chapter 5, for video transmission, it could be interesting to define an action that implements to change streaming parameters. In order to per-



1 Byte	1 Byte	2 Bytes
Type = 1	Len = 18	Obj ID

1 Byte	1 Byte	2 Bytes
Obj Type	Losses	Delay

2 Bytes	2 Bytes
Jitter	Pad

8 Bytes
Bandwitdh

Figure 4.32: RTG_SET_ACTIONS message.

form that action, the system has to use the API of a video service northbound application.

4.4 Conclusion

In this chapter, two different approaches to implement SDN-based routing have been proposed. The first one is an adaptation of the traditional routing algorithm OSPF, achieving an increment of performance. However, that solution is limited to the previous routing implementation.

In order to take full advantage of the SDN architecture, a new protocol was proposed. This protocol defines the states, the data structures, the algorithm and the messages needed to work. It uses the SDN controller capabilities to implement routing in the network. Furthermore, an AI module is defined in the architecture. The routing module uses this AI module to take decisions based on intelligent algorithms. Both the AI module and the routing module can be used as external applications based on the SDN architecture, through the Northbound API.

In chapter 5, the AI module is described. All its functions and the intelligent methods it is based on are explained. An intelligent method to choose the

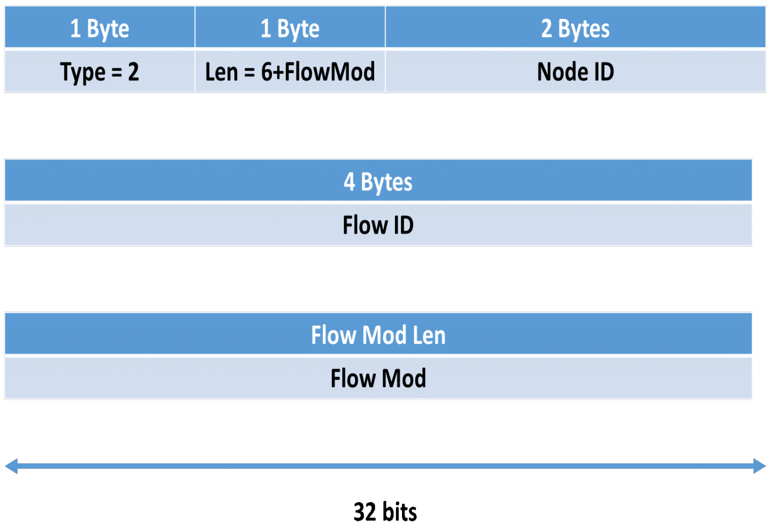


Figure 4.33: RTG_FLOW message.

most adequate action will be discussed. Furthermore, the whole proposal is tested.

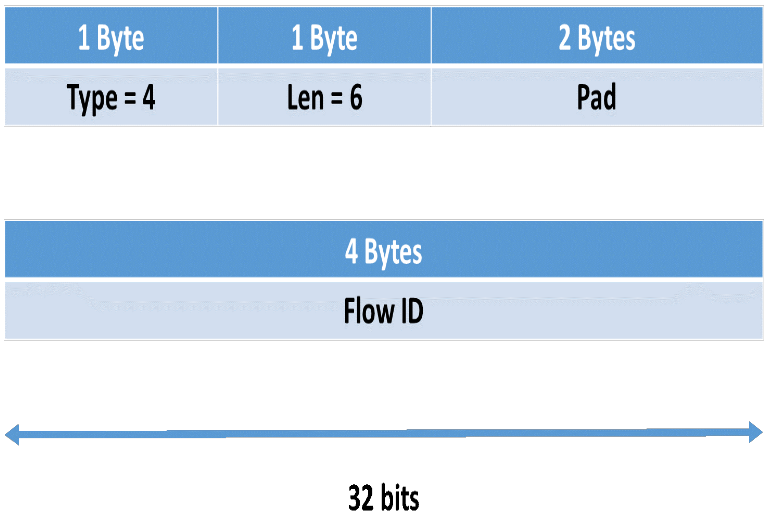


Figure 4.34: RTG_MULTIMEDIA message.

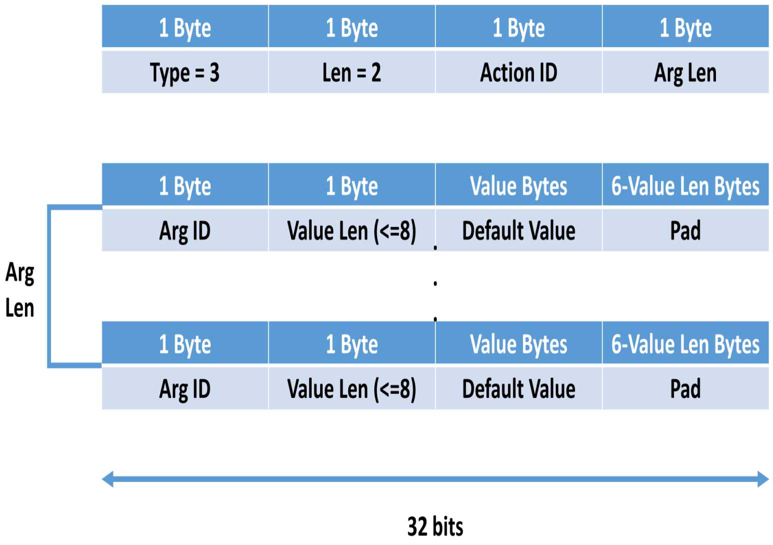


Figure 4.35: RTG_PERFORM_ACTION" message.

Chapter 5

Artificial Intelligent Module

5.1 Introduction and architecture

In this chapter, the artificial intelligence module that was introduced in chapter 4 is described. In the architecture, the SDN controller uses two different modules, the AI module and the routing module, which cooperate to provide routing to the network.

In chapter 4, the routing module and its tasks were described. The routing module was in charge of analyzing the topology, notifying the flows of the network, providing the QoS parameters of the nodes, flows and routes in the network.

With all these functions implemented by the routing module, the functionalities that the AI module has to implement are the following ones:

- Multimedia flows identification
- QoS/QoE degradation estimation
- Choice of the most adequate action
- Evaluation of the impact the action had in the network

The first task is not in the scope of this work. The multimedia flows identification will be based on the multimedia protocols and the TCP/UDP ports the flows use. In this chapter, the second and third tasks are explained. First, two different studies about the parameters that are related with the QoS and QoE are detailed. The first one is a video transmission study. The second one will be the model used to detect QoE degradation in the system. Once the QoE degradation estimator is explained, the artificial system in charge of the decisions in the routing protocol is discussed. After the system is explained, the evaluation of the impact that the action has in the network will be discussed. Finally, the chapter ends with a comparison test between different routing proposals for multimedia transmission routing in SDN.

5.2 Video Transmission Study

In this section, the study performed to design the AI system is presented. Mininet emulator has been used to perform the simulations and gather the data to design the system. The network emulated is based on the architecture shown in Fig. 5.1. With this testbed, the goal is to obtain the most complete set of data. We need to check which characteristics have influence over the QoS obtained, and consequently over the QoE. Therefore, we have analyzed how the video resolution and the frame rate affect some network parameters. These network parameters are jitter, delay and loss rate. Moreover, we have also checked how the network status affects the video quality perceived by the end user.

In order to perform the study, we have used several compression formats and different networks. The video formats used have been MJPEG, MPEG4 and H264, with different video resolutions. The video resolutions have been chosen based on the fact that IP surveillance video cameras use sensors from 0.4MP (Megapixels) to 8MP. Some of them reach higher video resolutions like 704x576 or UHD. Therefore, we have used three types of resolution for testing: a medium resolution, a high resolution and a very high resolution. These resolutions are 928x576, 2592x1920 and 3840x2160, respectively. The number of frames per second is also taken into account. Rates between 15fps and 30fps are usual in these kinds of cameras. The networks used in the test have been both wired and wireless.

Once the different scenarios and video characteristics have been set, the simulations are executed, and the data are gathered. On each simulation, the network parameters (jitter, delay, loss rate and bandwidth) of the links of each node are captured. In the end, the data is collected, and the dataset is built. This dataset brings us the possibility of testing our system once it is developed. So, the performance of the AI system can be measured.

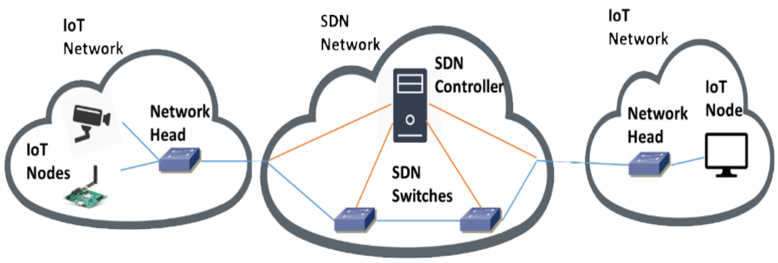


Figure 5.1: Elements and network of the proposed architecture.

5.2.1 Development and Analysis of QoS Results in Video Transmission

In this subsection, the results obtained from the testbed to build our dataset are shown and analyzed. In order to analyze the maximum possible number of scenarios for video transmission under the system proposed, we executed several tests that can be classified into:

- Congested networks
- Networks with high loss rate
- Networks with jitter

Each one of these cases is analyzed.

First, we have analyzed how different frame rates (30-60 fps) and resolutions (800x600, 1280x720 and 1920x180) affect to jitter, delay and loss rate. This test has been performed on a wired network with loss rates from 0% to 1%.

Figs. 5.2-5.7 show the results of this test. It shows how loss rate affects the QoS parameters when the video resolution changes. Delay (Fig. 5.2), lost packets, average lost packets and jitter are measured. The number of lost packets is shown in Fig. 5.3. As displayed in Fig. 5.4, the losses obtained are higher when the video resolution is low, reaching more than 20 packets as maximum, as can be observed in Fig. 5.3. This pattern is also presented with jitter and delay, shown in Figs. 5.2-5.5. In addition, QoS is degraded at the beginning of the transmission because of the increment of jitter, delay and loss rates. This increment is the highest increment of the transmission. The increments occur at 8s, at 41s and also at 45s, when there are changes on the video scene. The delay increases up to 1.7ms of maximum and, during the rest of the transmission, it varies from 0.2ms to 0.3ms. Then, at the end of the transmission and, as shown in Fig. 5.2, it increases again up to 1.4ms. The number of lost packets presents a maximum of 23 packets at the beginning of the transmission and then decreases to 4-5 packets. At the end of the streaming, it raises to 24 packets for an 800x600 video resolution and a 0.5% loss rate (Fig. 5.3). Regarding jitter, it reaches 3ms at the beginning and 1.8 at the end (Fig. 5.5). In Fig. 5.6, the manner in which the loss rate affects the delay when a 1920x1080 30fps video is being streamed can be observed. The highest changes on delay happen when the video scene changes. Delay increases to 0.5ms at the beginning and to 0.65ms at the end of the streaming. The loss rate presented in this test is not very significant because the different measures with 0% and 1% of loss rate are similar. In order to show how frame rate affects transmission, Fig. 5.7 is presented. The highest the frame rate gives the highest delay. The maximum delay is 4ms at the end of the transmission with 0.2% loss rate. The peaks of delay happen on different timestamps due to the different video framerate.

Secondly, we have analyzed how different values of jitter, from 10ms and 60ms, affect the QoS parameters under streaming with different rates (30-60fps) and resolutions (800x600, 1280x720 and 1920x180). The results are displayed in Figs.5.8 -5.11.

In Fig. 5.8, we can observe how the increment of jitter affects the delay on transmission with different resolutions. Again, when the video changes the scene, the increment of delay is higher. It is incrementally increased to 0.6ms at the beginning and 0.9ms at the end of the 800x600 video transmission. The higher the resolution, the lesser the delay is. With the same video resolution, the increment of jitter does not affect the delay. However, in Fig. 5.9, we can observe that, with higher resolutions, we obtain more lost packets. With 1920x1080 video resolution, the maximum number of lost packets is 9 at the beginning of the transmission. Nevertheless, with an 800x600 video resolution, we obtain 16 lost packets of maximums at the end. The peaks of loss are placed in the last two changes of video scenes, on 41s and 45s. In Fig. 5.10, we have the same results in terms of average lost packets. We observe the same pattern, with increments at the end of the streaming. In Fig. 5.11 we observe that jitter is affected in the same way as delay. We obtain lower values, with maximums

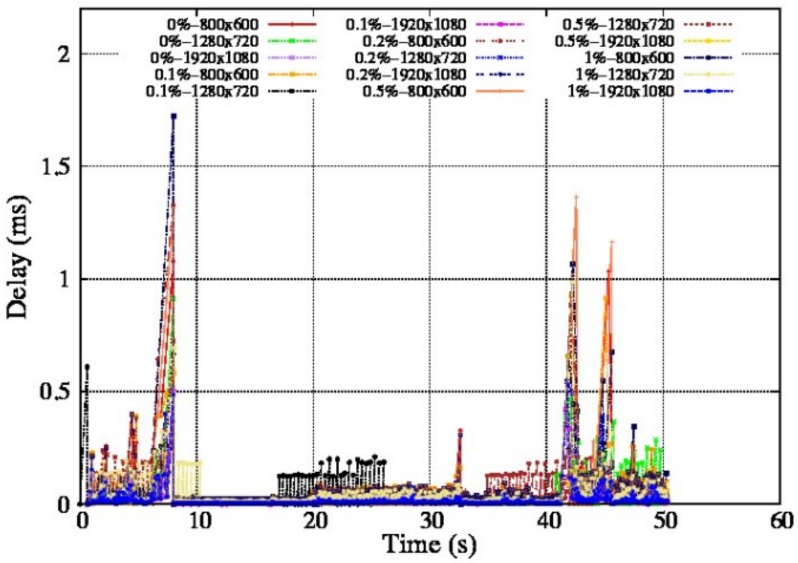


Figure 5.2: Delay with loss rate from 0% to 1% and different resolutions with 30fps.

of 0.3ms at the beginning of the 800x600 transmission and 0.8-0.9ms at the end of it.

Finally, we have analyzed how QoS parameters vary when there are different videos with different rates and resolutions. These videos are sent through networks with different bandwidth from 10Mb/s to 100Mb/s and causing congestions. In Fig. 5.12 and Fig. 5.13, the delay obtained for different transmissions through a path with 10Mb/s and 100Mb/s of available bandwidth is depicted. In order to congest the network, an 800x600 30fps “disturbing” video is streamed as displayed in Fig. 5.12, and a 1600x1200 30fps video is streamed in as displayed in Fig. 5.13. They started at different moments. This process is repeated after 30 and 40 seconds of transmission. As we can observe in Fig. 5.12, the delay obtained is higher for videos with lower resolutions. For instance, with 800x600 streaming, the delay reaches 1.7ms of maximum at the beginning of the transmission, but with 1024x768, it is lower than 1.6ms. Moreover, as it can be observed in Fig. 5.13, the delay increases up to 1.1ms at the end. We have repeated the process, increasing the available bandwidth to

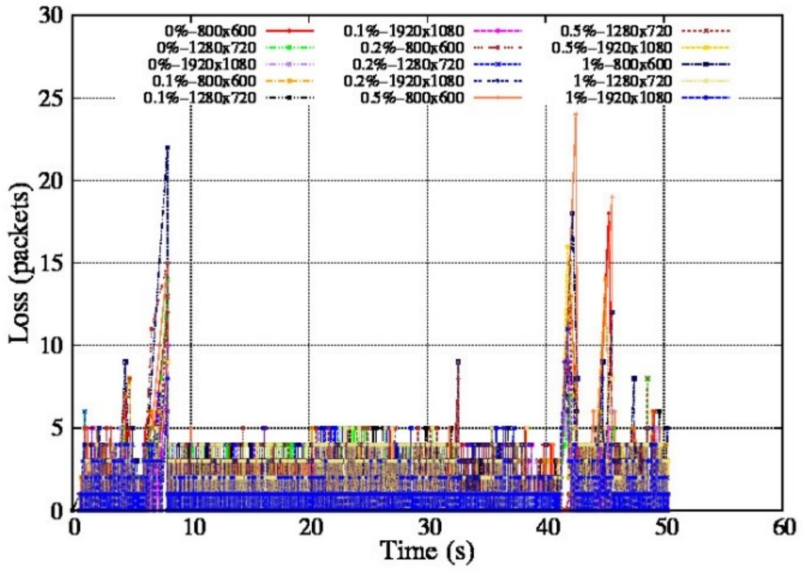


Figure 5.3: Number of lost packets with different resolutions with 30fps.

100Mb/s. In Fig. 5.14, we observe that this increment of available bandwidth reduces the impact on delay, being 1.3ms during the beginning of the 1024x768 transmission. In Fig. 5.15, it is shown how changes in the number of lost packets occur when the 10Mb/s network gets congested. The video transmitted has an 800x600 resolution and 30fps. The video is transmitted at 10s(s1), 20s(s2) and 40s(s3). When the 1920x1080 video is being streamed, there are losses of 50 packets. In Fig. 5.15, we can observe a frame of the video when the peak of lost packets happens. That frame would decrease the QoE obtained. This error happens at 48s.

Fig. 5.16 shows that there is not a relationship between loss rate and the size of the video, but there is a relationship with the video resolution. The transmission, in terms of packet loss, is more affected with less video resolution. This can be observed in the red, green and orange lines in the graph. These lines are the ones related to 800x600 video streaming. However, the errors produced when a high-resolution video is being streamed affects the QoE more, as we can observe in the 1920x1080 (s2) case. If the network is congested with

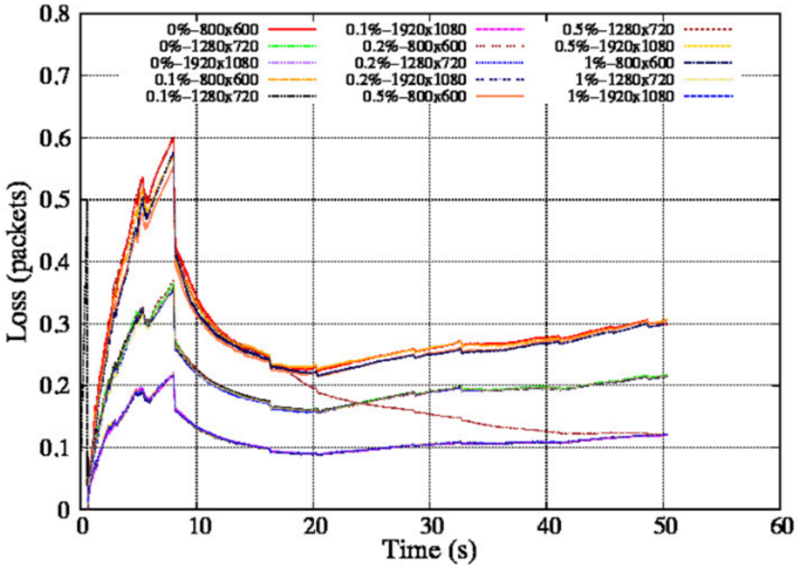


Figure 5.4: Average of lost packets with different resolutions with 30fps.

a 1024x768 video, we obtain the results that are presented in Fig. 5.17. With these results, when the resolution is low, the transmission is more sensitive to loss rate. This can be observed in the pink, orange and blue lines for 1024x768, and these results are compared with the lines related to 1920x1080. However, this does not affect the MOS, as we have observed when analyzing the received videos.

5.2.2 Development and Analysis of QoE Results in Video Transmission

We have performed 495 experiments. On each one, we have saved the video received on the destination. Thereby, we have both, the original video and the video with transmission errors. So, we can perform an objective study of the image quality received. As a result of this study, we have obtained PSNR, NQI, VQM, SSIM and MSE measurements. The goal was to obtain an approximation of the level of quality perceived by the user. Attending to

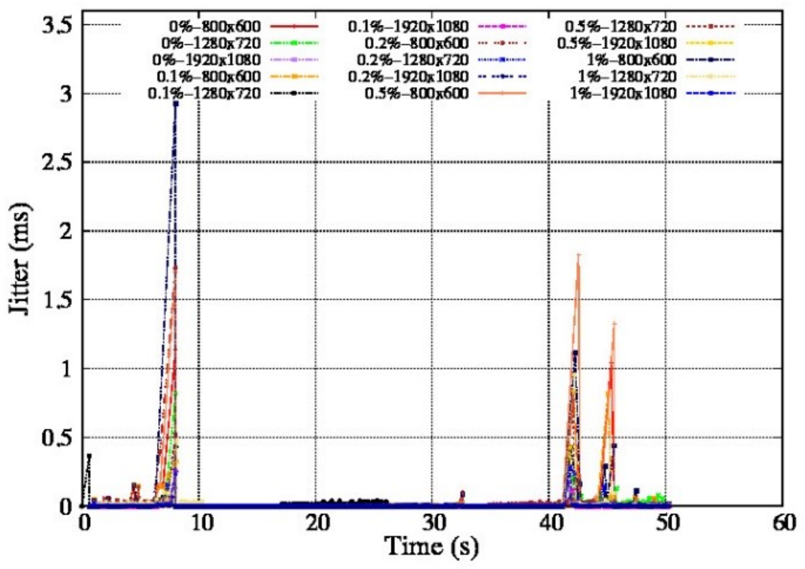


Figure 5.5: Jitter in ms with different resolutions with 30fps.

the literature, we have chosen those measurements that have more correlation with the subjective quality, in terms of MOS, perceived by the user.

The Peak Signal to Noise Ratio (PSNR) parameter is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Normally, higher PSNR indicates that the reconstruction is of higher quality [170], [171]. According to the mathematical equations for calculating MSE and PSNR, it can be inferred that they represent similar error values (i.e., the calculated error is of the same degree). Because of this, PSNR can be considered as an unofficial representative of all the above-mentioned video quality metrics and still the most widely used metric for video quality estimation in many video processing systems [172]. The Human Visual System (HVS) is highly adapted to extract the structural information from the area of viewing. SSIM metric uses this characteristic of the HVS in the estimation of the quality of the processed digital video. Structural information of an image can be defined by those characteristics that represent the structure of the objects

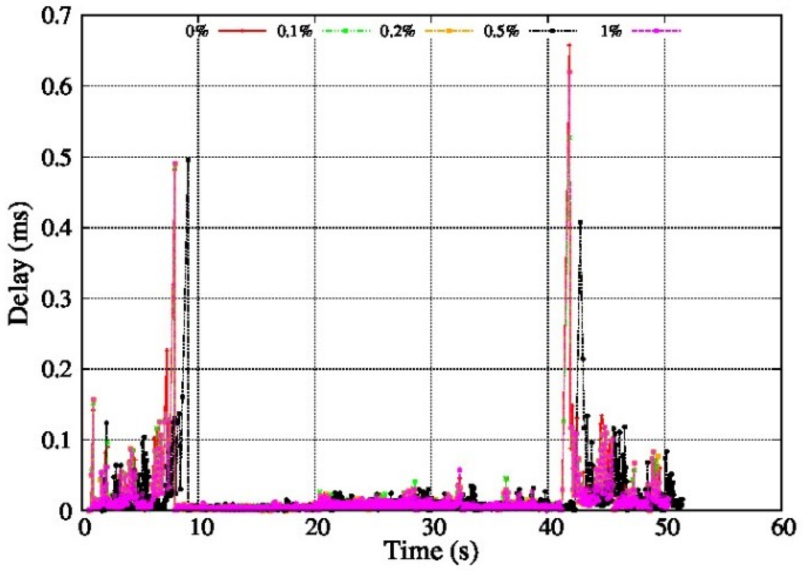


Figure 5.6: Delay with loss rate from 0% to 1% and a 1920x1080 30fps video streaming.

in the scene – independently of the mean brightness and contrast [172-173]. These measurements are based on three components: luminance comparison, contrast comparison and structure comparison [75]. NQI works in a similar manner as the SSIM index. NQI defines picture distortion as a combination of three factors: difference in mutual characteristics, difference in luminance, and difference in contrast. Human eye sensitivity to spatial-temporal patterns decreases with high spatial and temporal frequency. Based on this difference in sensitivity, high spatial or temporal information can be represented with less data and less precision, while human eyes are more or less insensitive to the loss of this information. This characteristic of HVS is exploited by DCT quantization, which is the base for VQM [170]. The values of VQM start from 0 and, in real situations, they can reach around 12. The VQM value of 0 represents minimum distortion and maximum quality [75]. In conclusion, the SSIM metric has a quite better performance compared to PSNR and, in most cases, performs very similarly to the Human Visual System. But imperfections are also present. SSIM is almost insensitive to changes in brightness, contrast and hue such that, when these changes are bigger, SSIM values can become

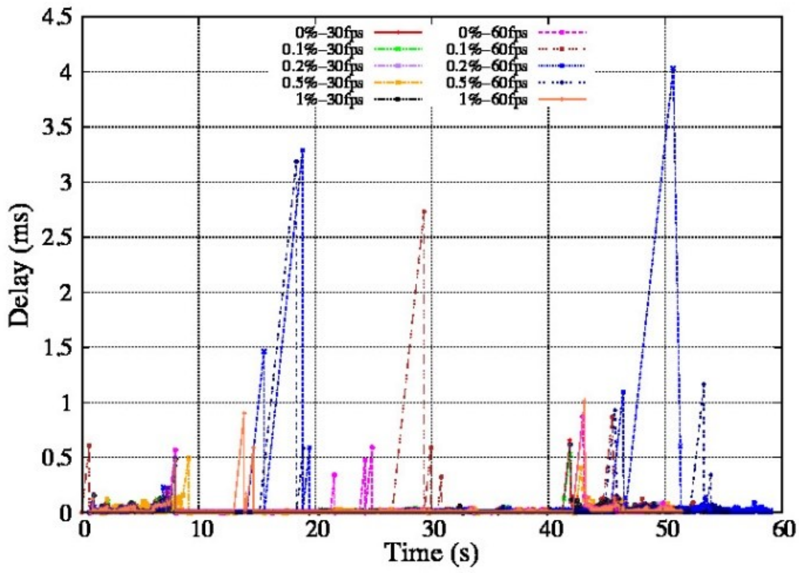


Figure 5.7: Delay in the same conditions as the previous case but also with 60fps videos.

largely inverted. VQM mostly considers the changes that are more noticeable to the human eye.

With all the previous discussion, we have chosen VQM, SSIM and PSNR to set the output parameters of the dataset. We have chosen these metrics because every one of them can provide us some kind of characteristic to correlate that metric with the QoE. VQM is similar to the subjective quality perceived by the user. Therefore, this metric has more weight in the settings. The equation used to calculate the subjective values from objective measurements of each frame is:

$$QoE_{\approx s_i} = \delta_{i,VQM} R_{i,VQM} + \delta_{i,SSIM} R_{i,SSIM} + \delta_{i,PSNR} R_{i,PSNR} \tag{5.1}$$

Where the subjective approximate QoE value for each frame i, $QoE_{\approx s_i}$, is the combination of three factors. The result of each metric M for the frame i ($R_{i,M}$) adjusted with a specific weight $\delta_{i,M}$. These values correspond with

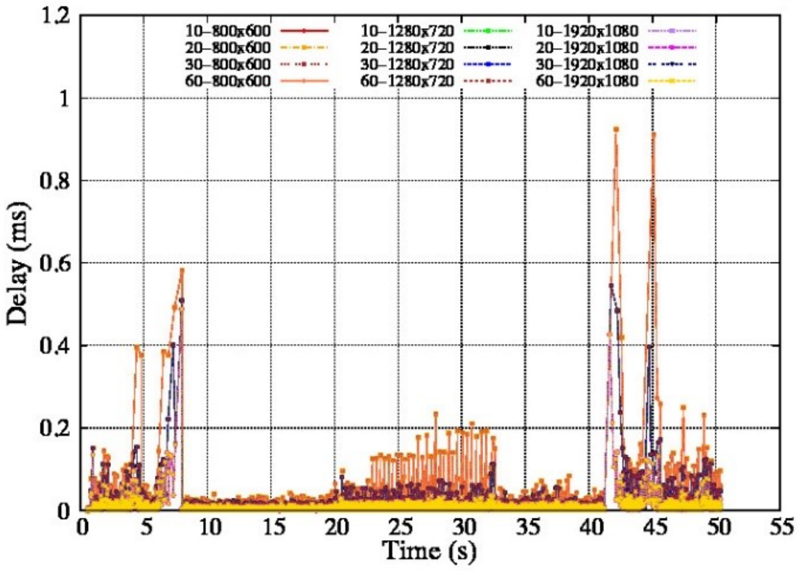


Figure 5.8: Delay depending on jitter, from 10ms to 60ms, and video resolution with 30fps streaming.

$\delta_{i,VQM} = 0.5$, $\delta_{i,SSIM} = 0.35$ and $\delta_{i,PSNR} = 0.15$, which are determined based on the literature.

5.2.3 Data Preprocessing

In order to set the data used in the traffic classifying model, we first performed data preprocessing. Thereby, we grouped the data in GOPs of 2 seconds. That means that if the obtained data was frame-based, we would preprocess them each 60 frames when the frame rate is 30fps, or each 120 frames when it is 60fps. Results are presented in Figs. 5.18 – 5.21.

As we can observe in Fig. 5.18, we converted the approximated 3000frames for 60fps videos and 1500 frames for 30fps videos into 25 samples as shown in Fig. 5.19. Each sample was formed by calculating the average of the values within the 2-second interval. The same process has been executed for the data packets. In that case, they are presented depending on the time instant, not

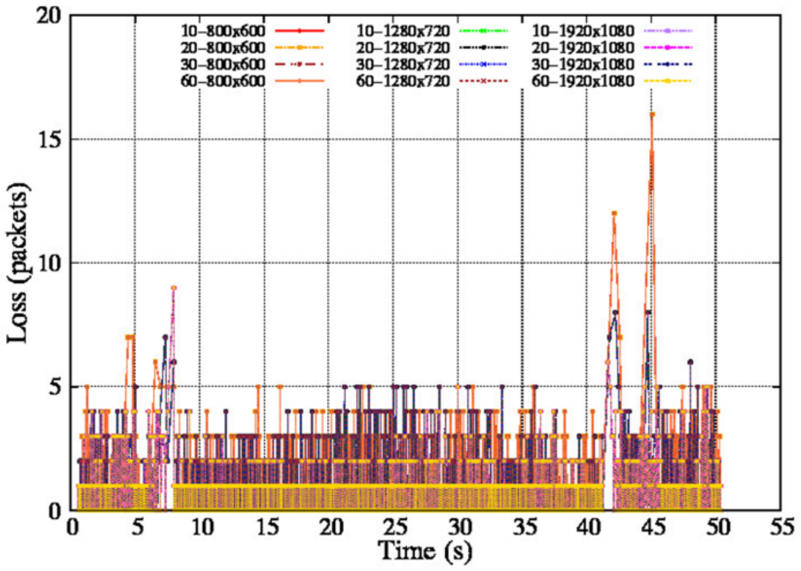


Figure 5.9: Number of lost packets depending on jitter, from 10ms to 60ms, and video resolution with 30fps streaming.

frames. As it can be observed in Fig. 5.20 and Fig. 5.21, the average data was taken every 2 seconds in order to obtain 25 samples from 50 seconds of transmission. Each sample corresponded to 2 seconds of video. In those 2 seconds, we obtained the average of different QoS parameters like jitter and delay (Fig. 5.20) or number of lost packets (Fig. 5.21). Regarding lost packets, there was a peak with more than 1.6 packets at the beginning of the transmission and another with 0.6 at the end. However, during the transmission, the average was below 0.2 packets. As in the previous study, these peaks happen when the video changes the scene. In terms of delay, at 8s, it reached 0.06ms and, at 42s, it reached 0.02ms. The average during the transmission was below 0.01ms. The jitter suffered an increment of 0.01ms at 8s, but it was insignificant during the rest of the transmission.

Once the preprocessing was done, we obtained 25 samples for each experiment. Jitter, delay, lost packets, bandwidth, resolution and fps were measured. The average value of the different objective metrics for each sample and time inter-

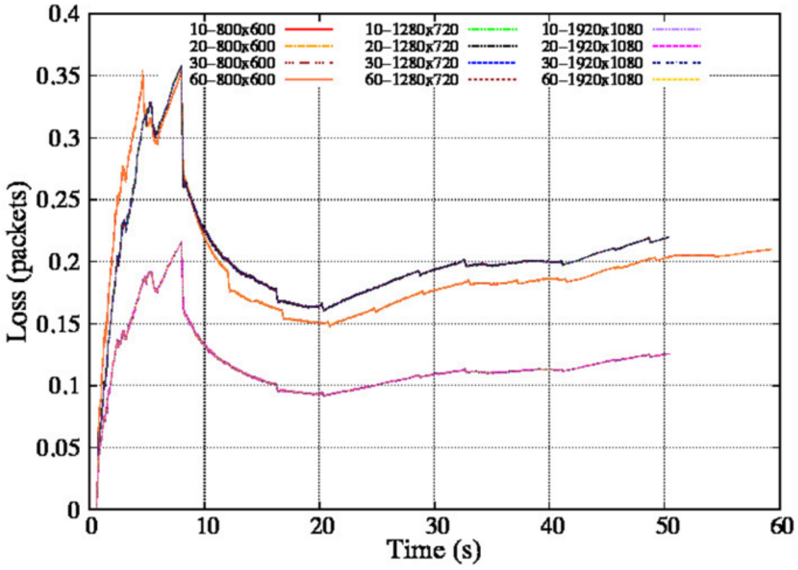


Figure 5.10: Average of lost packets depending on jitter, from 10ms to 60ms, and video resolution with 30fps streaming.

val was used for labeling. Consequently, we obtained 6 QoS-or-video characteristics for each 2 seconds as an input for our system. As output, we obtained the label that corresponded to the average of the objective QoE metrics. The last step in the preprocessing process was to divide the spectrum of the possible objective QoE values into 5 ranks. Thereby, the labels used were discrete values, not continuous. Managing only 5 possible labels is much easier. We associate the kind of traffic to those labels as we describe in Section 5.3.

5.3 QoS Degradation Estimator

In this section, how the development of the multimedia traffic classification model was carried out is explained. This classificatory determines the estimation of the QoS degradation of the multimedia transmission. First, a QoE classification model is developed. Then, according to the results, we have

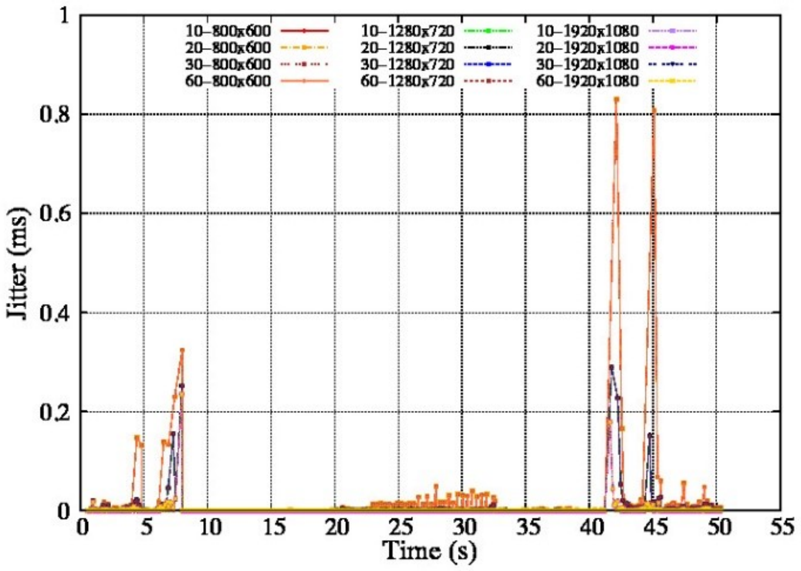


Figure 5.11: Jitter depending on jitter, from 10ms to 60ms, and video resolution with 30fps streaming.

analyzed and evaluated which model works better for the multimedia traffic classification.

5.3.1 Database Pre-processing

The database used in this subsection to develop the intelligent multimedia traffic classification system comes from the work done in [74]. It is composed by 2741 samples with 14 features per sample. Each sample is labeled according to the patterns obtained. Moreover, their values are obtained from objective QoE measurements, which have a high correlation with the subjective QoE, like ISSM, VQM and NQI. PSNR is also used due to its importance for the study. In order to extract every single sample, several video transmissions with different network and video conditions were sent. Each video transmitted were divided into GOPs (group of pictures) equivalents to a 2s video clip. The average values of some characteristics were extracted from each video clip. Some of these features are: frame count, stream size, proportion stream, footer

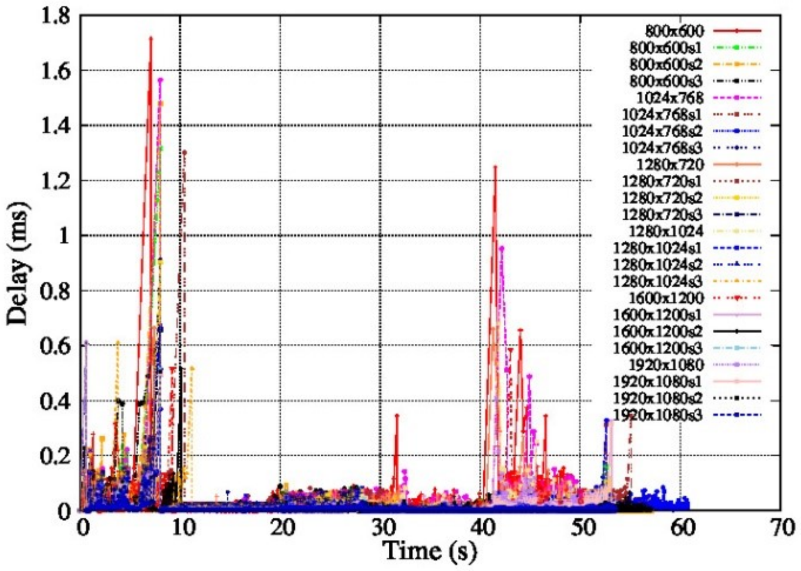


Figure 5.12: Delay obtained in a 10Mb/s network with 800x600 30fps disturbing traffic.

size, bitrate, width, height, frame rate and minimum frame rate. Furthermore, some network parameters like delay, jitter, packet loss, jitter variation and bandwidth for each time interval were extracted. For each video clip, several objective measurements were extracted by comparing the source and the destination videos so that the class labels could be generated. Depending on the correlation with the subjective QoE, a weight was assigned to each one of them and an average was calculated.

5.3.2 QoE Estimation Model

An important aspect of this study is to obtain an accurate QoE estimation model with which a high-reliable multimedia traffic management system could be developed. In [74] only a classification model of the type of traffic was obtained. This classification model is addressed in Subsection 5.3.3. In this subsection, a wide study about QoE estimation is firstly performed to improve the results of the traffic classification model. For this study, several classification and regression algorithms have been evaluated. The problem can be

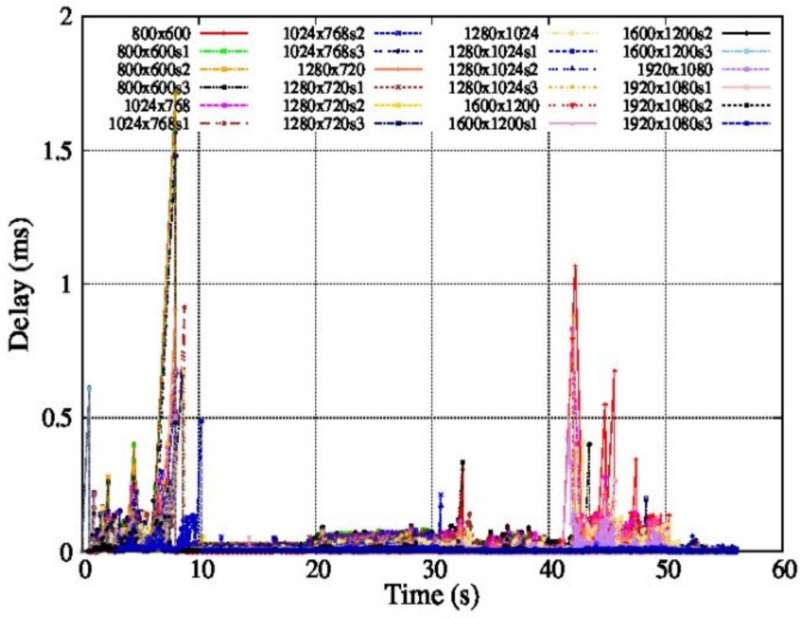


Figure 5.13: Delay obtained in a 100Mb/s network with 1920x1080 30fps disturbing traffic.

addressed from two different points of view. It could be either a classification problem where discrete (1:1:5) MOS values are obtained as results or a regression problem where the result is a set of continuous values (1, 5). In this work, the problem has been turned into a classification problem by transforming continuous MOS values into discrete values. This means, the output continuous values of the regression model have been rounded to discrete values. Therefore, the regression problem changes into a classification problem. This allows using regression models and check how good they suit the problem. In order to obtain the models, the following algorithms, classification models, have been used:

- Support vector machines (CSVM)
- Neural networks (CNN)
- Naïve Bayes (CNB)

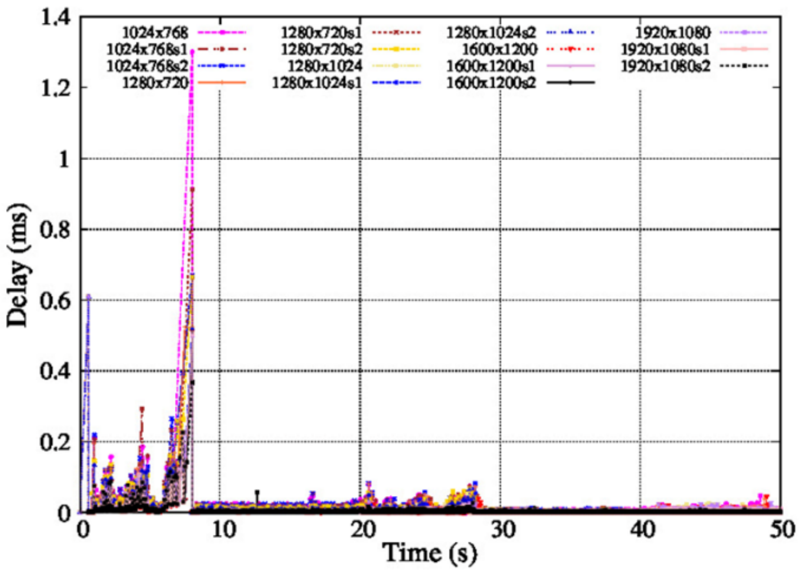


Figure 5.14: Delay obtained in a 100Mb/s network with 1920x1080 30fps disturbing.

- Decision trees (CDT)
- Discriminant analysis (CDA)
- K- Nearest neighbors (CKNN)
- Regression models based on:
- Gaussian process (RGP)
- Non-linear (RNL)
- Decision trees (RDT)
- Neural networks (RNN)

In order to carry out this study, MATLAB and the available machine learning libraries have been used. The array of features of each sample has been used as entry parameters in the supervised learning models, and the MOS is used

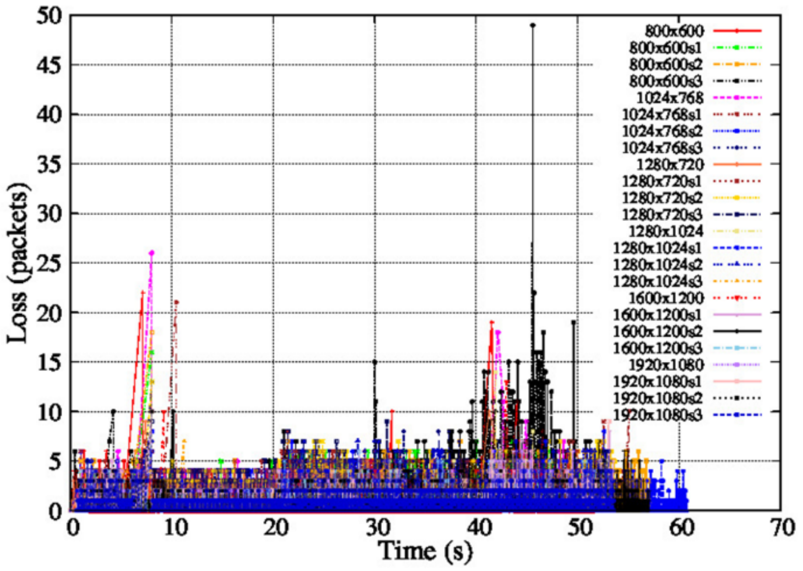


Figure 5.15: Number of packets lost in a 10Mb/s network with 800x600 30fps disturbing traffic.

as the class label. From the whole simple set, 2441 samples have been used for training and 300 for testing. Cross-validation has been used to avoid the overfitting and, thus, generalize and validate the models. Concretely, 10-fold cross-validation have been used.

The procedure of model obtaining has been as follows. Firstly, the different models have been evaluated using the default parameters. Then, the models with the best classification results have been polished to find a local or global minimum that improves the classification results. Several scripts have been created to analyze and evaluate several learning parameters like kernel function, optimization coefficients, stopping criteria, etc. In order to evaluate the classification results of each proposed automatic supervised learning algorithm, the confusion matrix has been calculated. With the obtained values, the precision, recall, accuracy, F Score (F1) and proximity values have been calculated. These values are displayed in Fig. 5.22 and Fig. 5.23. Proximity has been included to obtain a less strict and rigorous accuracy measurement.

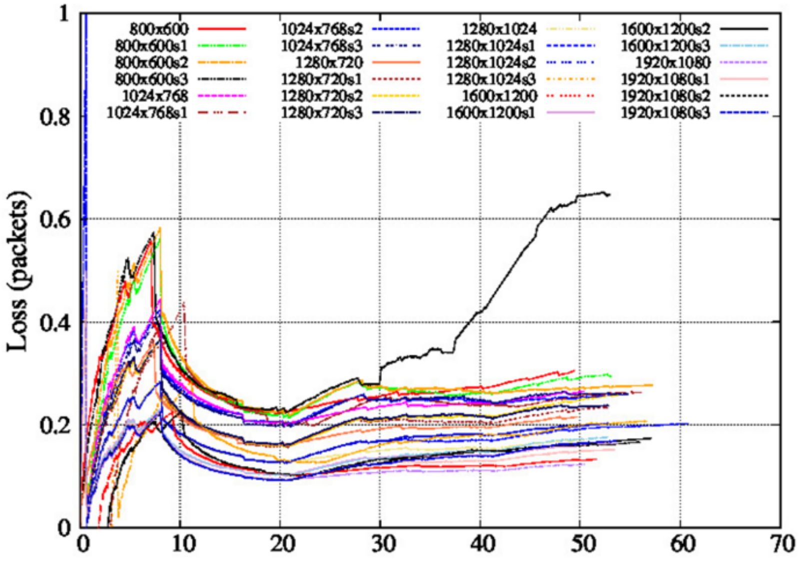


Figure 5.16: Average of lost packets obtained in a 10Mb/s network with 800x600 30fps disturbing traffic.

Precision indicates how good the system can be to detect the class that is being analyzed. This means, with regard to the samples of that class, how many has been labeled correctly. Proximity allows considering whether the system not only detects that class, but also if it is close to do it. This measurement gives an idea of how quasi-accurate the model is. It can be a big help to select the management system.

As it is depicted in Fig. 5.22, the first measurement is precision, which indicates the positive cases detected correctly. RNN and CSVM models are the ones with the best precision results. They have obtained similar results, with an 84%. Moreover, CKNN, CDT and RGP present good results with an 81% for CKNN and a 77% for CDT and RGP. Worse results are obtained from the models based on CNB and CDA with 44% and 65% respectively. These poor results for CNB and CDA are also obtained in the other measurements, except proximity, being unable to be greater than 46%. Since CNB presents good results with non-parametric data, it can be concluded that this is consequent

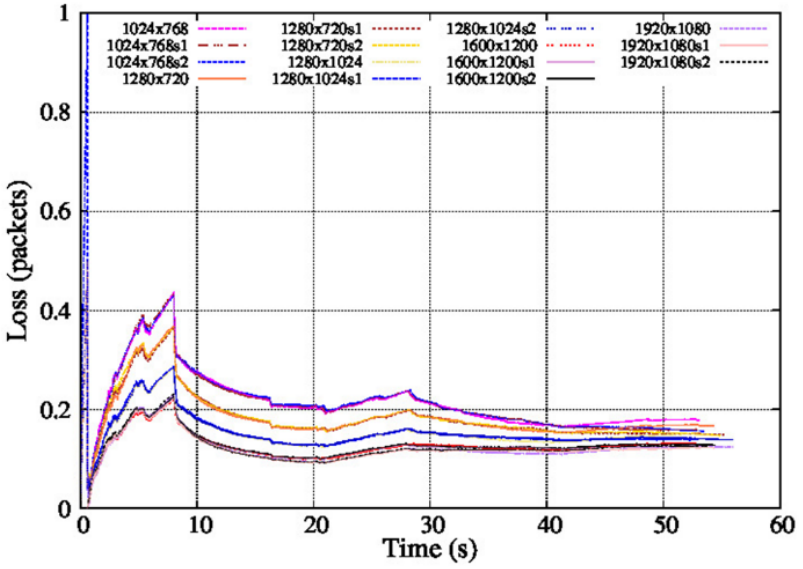


Figure 5.17: Average of lost packets obtained in a 10Mb/s network with 1920x1080 30fps disturbing traffic.

with the current problem and it is not an issue at all. As regards regression (Fig. 5.23), RDT and RGQ present the worst results, except for proximity, like in the previous case. This high value in proximity indicates that the model fails but it is close to be valid. On the other hand, regarding recall, i.e. the capacity of detecting properly the positive cases, RNN presents the best results with a 79%. CSVM and CKNN present similar results with 69% and 66% respectively. The next parameter, accuracy, shows how good the system is regarding the total correct estimations. CSVM presents the best results regarding accuracy (82%) followed by RNN (78%) and CKNN (72%). In this case, unlike the other measurements, CSVM presents a better result than RNN because all the right choices, from the whole set of possibilities (TP, TN, FP and FN), have been taken into account. This difference in the results only happen in those two scores. F1 has been another parameter tested. This parameter combines precision and recall in the same measurement to indicate the quality of the method. This score emphasizes the difference between the algorithms that presents good results and those which not. CSVM and RNN are the models

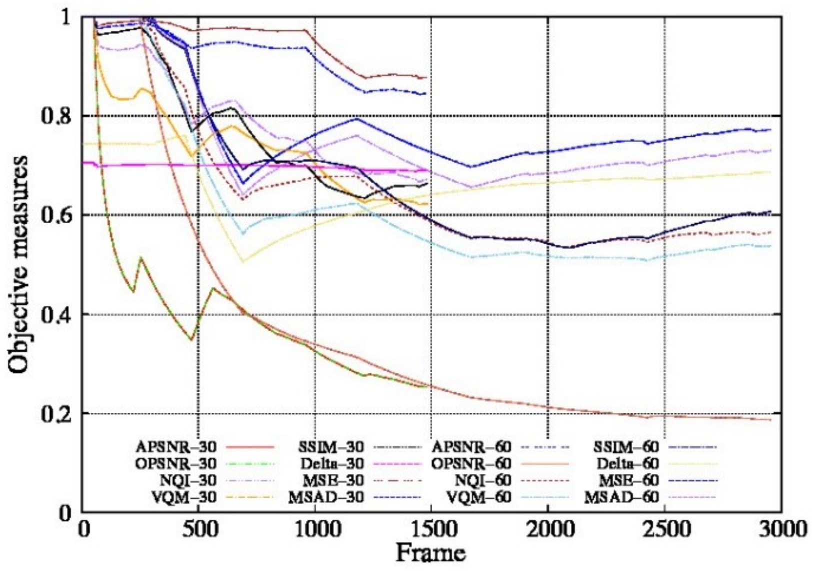


Figure 5.18: Dataset related to the subjective measurements after analyzing both, a 30fps and a 60fps video streaming.

with best results as regards this score. Finally, as regards proximity, RNN stands out with a 97%. However, the other regression methods achieve high results with 96%, 95% and 91% for RGP, RDT and RGQ respectively. This can be observed in Fig. 5.23. CSVM, CKNN and CDT present similar values, 93%, 92% and 91% respectively. This score is quite important for this work because it shows that if the model fails with the MOS prediction, it will be close. Therefore, it will not return a MOS of 5 when the actual MOS is 2, for instance. This aspect is really important to improve or polish the system.

As a conclusion of the results obtained from the confusion matrix, the most adequate models with regard to the problem are the ones based on CSVM and RNN. In Fig. 5.24, the behavior of both models in the test can be analyzed deeply. RNN has obtained less prediction errors than CSVM, facing the problem slightly better. When MOS = 5, both systems predict a right result, RNN with 93% and CSVM with 97% of right estimations. In the cases when RNN has failed, it has predicted MOS = 4 except in one of them, where it has

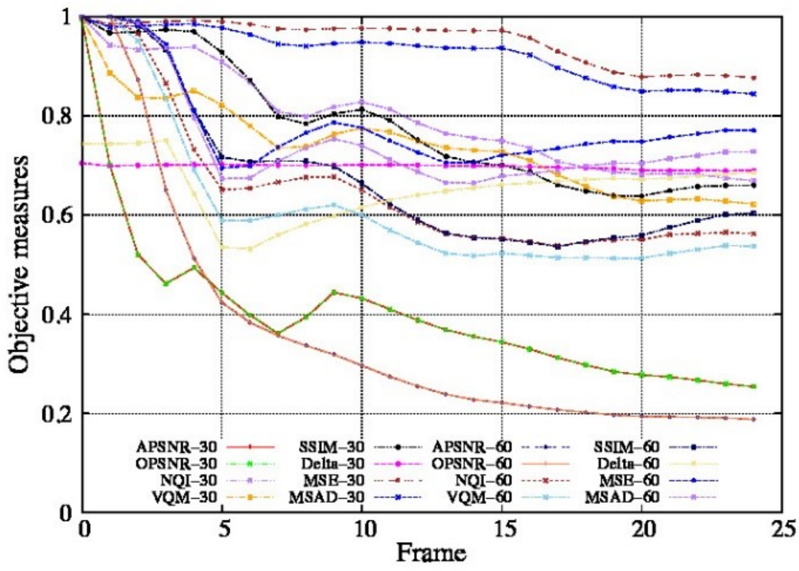


Figure 5.19: Preprocessing based on sampling for each frame.

predicted MOS = 3. CSVM has predicted MOS = 4 once and MOS = 2 in the other mistakes. When MOS = 4, CSVM predicts correctly in the 75% of cases, with MOS = 3 with 75% too and with MOS = 1 with 62%. It should be highlighted that, with MOS = 1, the number of errors increases significantly up to 14, a 38% of the total cases with MOS = 1. RNN predicts correctly when MOS = 4 with 70%, 77% when MOS = 3, with 78.5% when MOS = 2 and with 70% when MOS = 1. As it can be observed, the highest difference is located when MOS = 1, where RNN predicts better. It is a remarkable problem that CSVM predicts MOS = 4 and MOS = 5 with a 50% of probability in that case, when MOS presents an actual value of 1. These facts correspond to samples 3, 6, 8, 10, 14, 27 and 37. RNN predicts only in an 18% of times MOS = 5 (samples 27 and 37), while in the other times it predicts MOS = 1. With a MOS = 2, RNN only fails returning a predicted MOS greater than 3 twice (14.2%, samples 38 and 40). However, CSVM returns a predicted MOS greater than 3 in 9 of the 12 mistakes, a 75% of the total, in samples 2, 4, 5, 6, 40, 44, 45, 46 and 58. This result is really bad for CSVM because in all of these cases, the classification model described in the next subsection would not

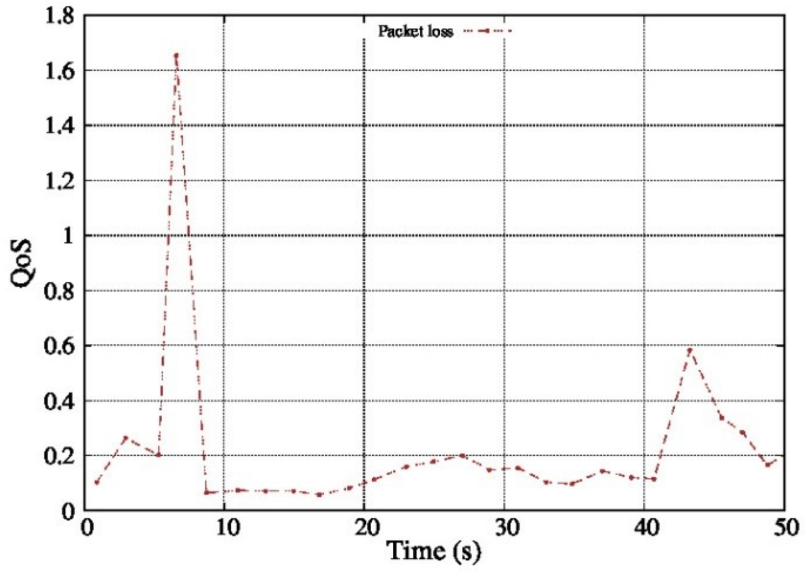


Figure 5.20: Preprocessing based on sampling the lost packets each time instant.

detect a critical traffic. When MOS = 3, RNN predicts a MOS greater than 3 in 6 of the 11 samples (4, 9, 26, 31, 42 and 43). CSVM predicts this in 14 of 21 (66.6%). Finally, when MOS = 4, the results obtained are quite similar for both models. With RNN 6 of the 16 samples (37.5%) were estimated with a MOS lower than 4 and with CSVM 3 of 13 (23%). Only the sample 32 was estimated with a MOS lower than 3 with RNN, while SVM estimated none with a MOS lower than 3.

In conclusion, the estimation levels of both models are quite similar. However, RNN presents better results because the wrong estimated values are not as far from the right ones as the presented by CSVM. This supports the results obtained from the proximity measurement. This fact is really important for the traffic classification, as it is explained in the next subsection.

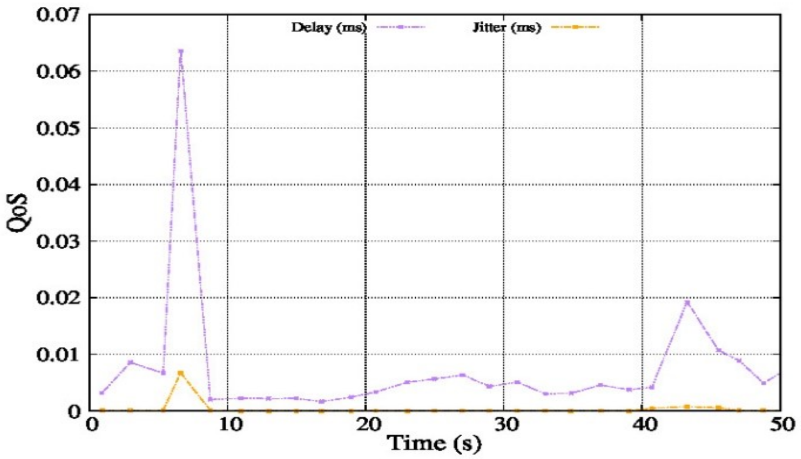


Figure 5.21: Preprocessing based on sampling jitter and delay each time instant.

5.3.3 Classification Model of the Type of Traffic

The classification model presented in this subsection is based on the proposal shown in [74]. This model classifies the type of traffic into critical or not critical. The traffic is considered critical when the MOS is less than or equal to 3.

This consideration was taken due to the subjective QoE, by watching a set of videos and checking when the perception of the quality started to be low. Since the videos were labeled according to the objective QoE, it could be checked that this subjective value matched a MOS of 3 in the model presented in [74]. In order to evaluate the different learning models presented in last subsection, the following criteria are used:

- TP (True-Positive): The model classifies the traffic as critical and it matches.
- TN (True-Negative): The model classifies the traffic as non-critical and it matches.
- FP (False-Positive): The model classifies the traffic as critical, but it does not match.

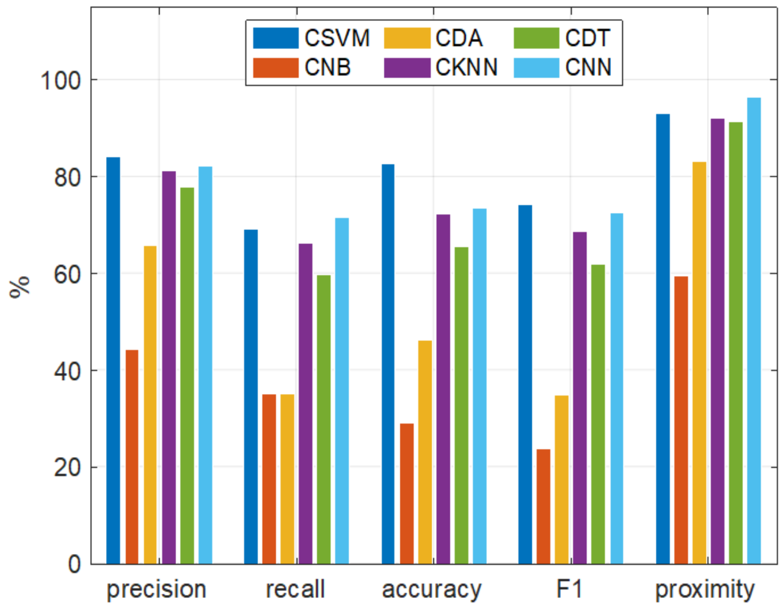


Figure 5.22: Learning results for QoE estimation in the different classification models according to the confusion matrix.

- FN (False-Negative): The model classifies the traffic as non-critical, but it is critical

After calculating the values of TP, TN, FP and FN, the precision, recall, accuracy, and F1 measurements of every model were obtained. Obtained results are depicted in Fig. 5.25 and Fig. 5.26.

In [74], CSVM and CKNN were the best methods with an 83% and a 73% of system accuracy respectively. In this work, these values have improved to a 93% and 83% respectively, as it can be seen in Fig. 5.25). It is also remarkable the result obtained from RNN with a 91% of accuracy (Fig. 5.26). In both, [74] and this work, the recall score presents the lowest results of all the measurements, except the 97% obtained in the statistic method in [74] and the 82% from the CNB (Fig. 5.25). Nevertheless, in [74] the statistic model did not work properly because it tended to extract a single value that matched a low MOS and matched the test samples. Therefore, that result was excluded. In this study, the same error happened. CNB tends to classify the traffic as

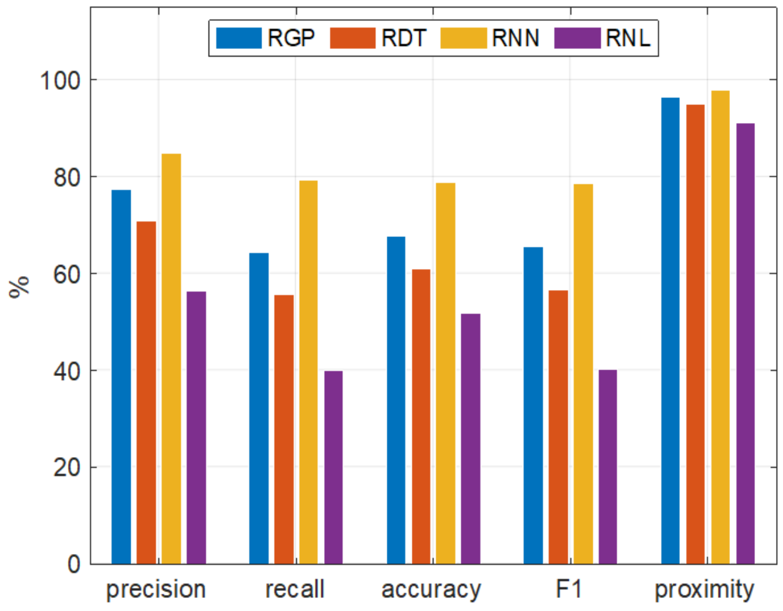


Figure 5.23: Learning results for QoE estimation in the different regression models according to the confusion matrix.

critical so that the 22% of the cases in which the traffic is actually critical it would classify it that way the 51% of the times. Thereby, and due to it low accuracy, the value of CNB is excluded. Then, the best value obtained is 86% from RNN. CNN also presents a good result of 80.6%. This measure is quite important for the system. Recall depends on FN, which means, the times the model classify the traffic as no critical when actually it is. Therefore, the number of learning samples should be increased to improve the test results in that way. This feature was already considered as future work in [74]. The fact that the result presented by CSVM, a 70%, is even lower than CKNN, with a 72%, is an important result. It can be expected, then, that CSVM will not be as efficient as RNN for the system. As regards accuracy, the best result obtained in [74] was 81% from CKNN. Although the result obtained from CKNN in this study is high (90%, the same result as RGP), it is lower than the ones presented by CSVM (92%) and RNN (95%). For the other models, the results are higher than 80% except for CNB (63%). The last analyzed score is F1, which indicates how good the models and the observations match. The

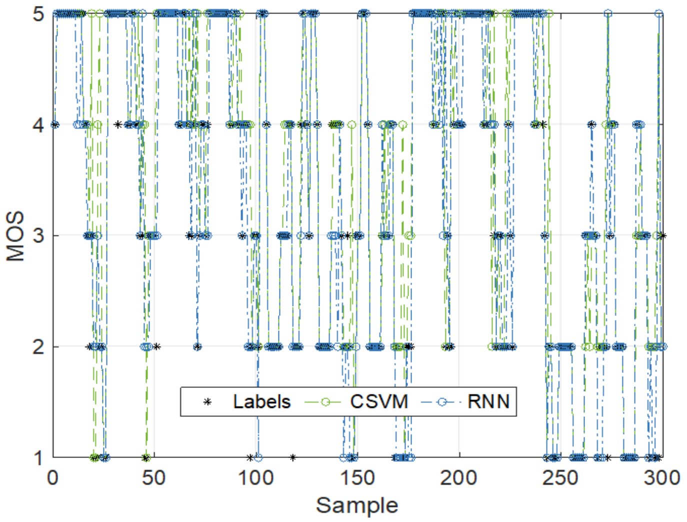


Figure 5.24: Test results of the QoE estimation for CSVM and RNN models.

best result obtained is an 88% from RNN, while the best one in [74] was the kernel-based function.

After the quantitative analysis of the different models, CSVM and RNN are the learning algorithms that present the best results, and they are the ones analyzed in the test. The result of the test is shown in Fig. 5.27, which shows that RNN presents a 5.6% of mistakes. A 41% of these mistakes have been made when the model has predicted the traffic as critical when it is not and 59% the other way round. Concretely, the model predicts that the traffic is critical and it is not in 7 samples (76, 122, 137, 138, 139, 140 and 192) and ten times the other way round (44, 67, 162, 193, 196, 217, 273, 288, 289 and 298). The classification errors increased for the CSVM model up to 11.3%. Unlike the other model, the majority of the errors in CSVM are produced when the algorithm classifies the traffic as no critical when it actually is. This happens in the 91.17% of times. This kind of error is more problematic due to the fact that the system would not handle several moments where the traffic transmitted is critical. Some of the samples of this case are: 17, 19, 22, 23, 97 and 225. On the other hand, the model has predicted only in 3 samples that the traffic is critical when it is not (137, 216 and 287).

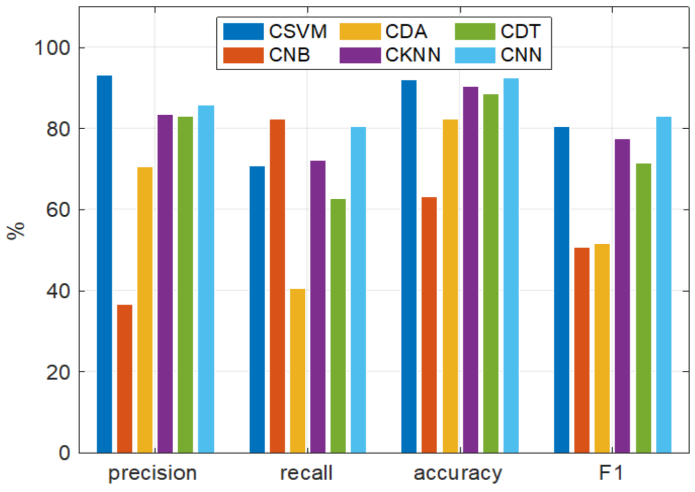


Figure 5.25: Training results for multimedia traffic classification obtained from the different classification models according to the confusion matrix.

5.3.4 Network Architecture of the Type of Traffic Classification Model

After the study, analysis and evaluation of the different automatic and supervised learning models proposed for type of multimedia traffic classification, RNN has been chosen as the most adequate. The optimization process of this neural evolutionary network modeled by a Bayesian regulation for critical multimedia traffic prediction is described as follows:

1. Gathering network data and video characteristics within different network environments and transmission characteristics to build the dataset.

2. Optimizing the NN architecture to find the most adequate number of the neurons in the hidden layer.

3. Peer data training application to optimize the NN architecture by using the Levenberg-Marquardt algorithm with the Bayesian regulation method until the network converges.

4. Introducing the test data into the well-trained and generalized NN to validate the predictive performance.

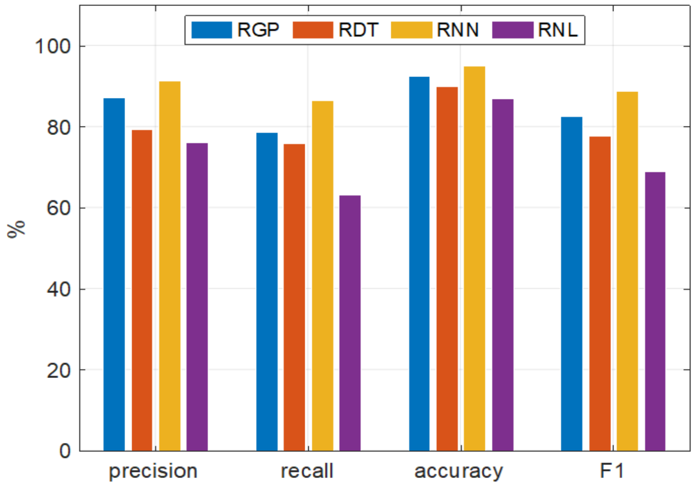


Figure 5.26: Training results for multimedia traffic classification obtained from the different regression models according to the confusion matrix.

The design of the network architecture result of these steps is presented in Fig. 5.28.

Where J is the number of neurons in the input layer, which corresponds to the set of features composed by network parameters and video characteristics. H is the number of neurons in the hidden layer and M the number of those in the output layer. This network is defined as:

Given n par of arrays

$$(\overline{x_l}, y_l) = (x_{l1}, x_{l2}, ..., x_{li} * y_l), \quad l = 1, 2...n \tag{5.2}$$

Where n is the number of samples and i the number of features of each sample. y_l uses discrete values in the interval (1:1:5). In order to calculate the values of y_l the following model has been used:

$$\hat{y_l} = \phi_0 \left(\alpha_0 + \sum_{h=1}^{H} \omega_{h0} \phi_h \left(\alpha_h + \sum_{j=1}^{J} \omega_{jh} x_{li} \right) \right) \tag{5.3}$$

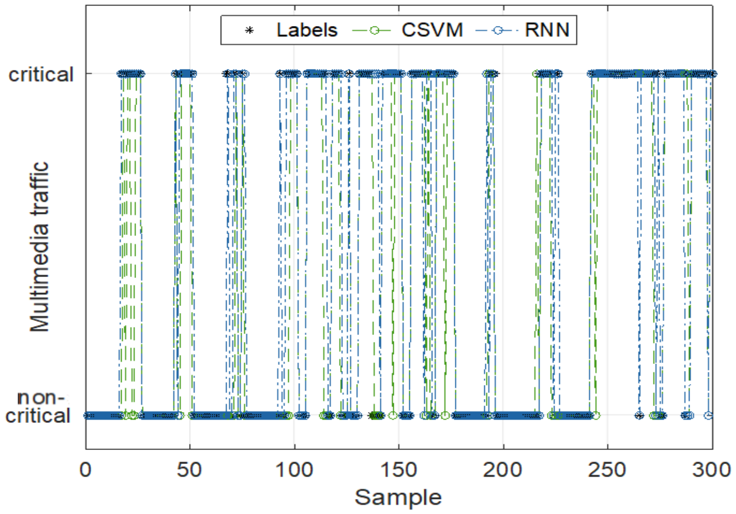


Figure 5.27: Test results of the multimedia traffic classification of CSVM and RNN models.

Where ϕ_h is the activation function of the hidden layer, ω_{jh} is the weight from the input layer to the hidden layer for the neurons between j and h, α_h is the bias value in the hidden layer. ω_{h0} and α_0 are the weights and bias from the hidden layer to the output layer and ϕ_0 is the transfer function from the hidden layer to the output layer.

The Bayesian regulation minimizes a lineal combination of squared errors and weights. Moreover, it modifies the lineal combination so that the network has good generalization qualities at the end of the training [174]. This regulation has been performed with the Levenberg-Marquardt algorithm. The different weights are calculated and adjusted within the learning process to minimize the error function E_D as follows:

$$E_D = \sum_{j=1}^{J}(y_j - t_j)^2 = \sum_{j=1}^{J} e_j^2 \tag{5.4}$$

Where J is the number of inputs of the data set D, and t is the label value.

In order to optimize E_D, the optimum values of the weights and biases have to been determined. One of the algorithms that solve this problem is Levenberg-Marquardt. This algorithm is faster than the backpropagation algorithm. It

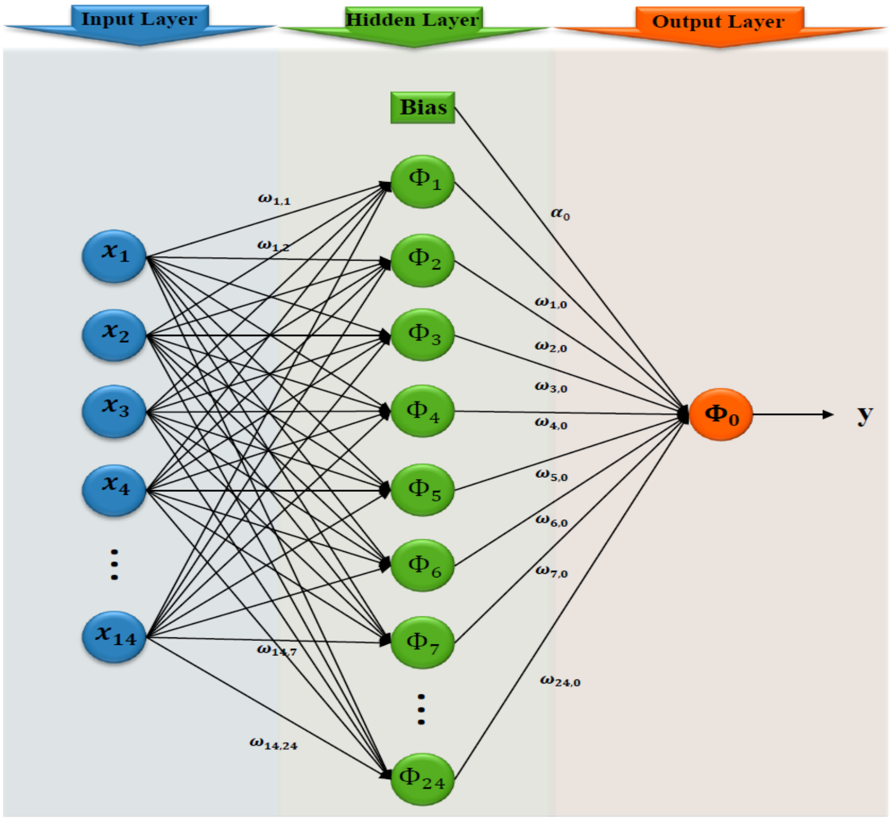


Figure 5.28: General architecture of the neuronal network used in this work.

has been designed to have a training speed close to second order algorithms. However, it has not to calculate the Hesse matrix directly, but it operates as it is explained next [175].

Hessian matrix (H) can be calculated approximately as:

$$H = J^T J \tag{5.5}$$

The gradient g could be calculated as:

$$g = J^T e \tag{5.6}$$

Where J is the Jacobian matrix that contains the first derivations of the network errors with regard to the weights and biases and e is an array of network errors. With backpropagation the Jacobian J^T is calculated and every variable is adjusted according to the Levenberg-Marquardt algorithm [176]. The Levenberg-Marquardt algorithm uses the Hessian matrix approximation with the following Newton algorithm variation:

$$\omega_{j+1} = \omega_j - (J_j^T J_j + \mu_j I)^{-1} J_j^T e_j \tag{5.7}$$

Where e is the error array, and I is the identity matrix. The adaptive value of μ is being incremented until the performance of the network starts reducing.

- The training stops when one of these conditions is true:
- The maximum number of repetitions is reached.
- The maximum time limit is exceeded.
- The performance is reduced up to the desired value.
- The gradient of the performance becomes lower than the minimum gradient.
- μ surpasses the maximum μ.

The optimization of the network architecture often takes a long time. The Levenberg-Marquardt algorithm is faster than any variation of the backpropagation algorithm [177, 178]. Levenberg-Marquardt is used to minimize the addition of the squared error and to overtake some of the limitations of the standard backpropagation algorithm like the overfitting problem [179]. Another different problem may happen if too few nodes are added to the hidden layer. If this happens the network will have problems in the learning process and the predicted results will be poor. In order to solve this problem, the BRNN algorithm incorporates the Bayes theorem in the regulation scheme. BRNN is, basically, a backpropagation network which combines the conventional addition of the minimal square error function with an additional term called "regulation". Therefore, from (5.4), the next equation is obtained:

$$S(\omega) = \beta E_D + \alpha E_W; \quad E_W = \sum_{j=1}^{m} \omega_j^2 \tag{5.8}$$

Where β and α are the regulation parameters and E_W is the penalty term, which penalizes big values for the weights, being m the number of weights of the system. In this context, the weights of the network are considered to be random variables. Then statistical techniques can be applied to calculate distribution parameters such as, for instance, variations. According to the Bayes' law, the distribution of probability can be written as [180]:

$$P(\omega|\alpha, \beta, D) = \frac{P(D|\omega, \beta)P(\omega|\alpha)}{P(D|\alpha, \beta)} \tag{5.9}$$

Where $P(\omega|\alpha)$ is the prior probability of the weights, which represents how reliable the weights are before gathering the data; $P(D|\omega, \beta)$ is the likelihood function, which represents the error probability; and $P(D|\alpha, \beta)$ is the normalization factor, named evidence in this model [181].

The optimal weights for the model can be obtained in the training phase by maximizing the posterior probability. It has the same results as minimizing the regulated objective function (5.8) [182]. If the weight and the distribution of the probability of the data are Gaussian, the prior probability of the weights can be written as follows [183]:

$$P(\omega|\alpha) = \frac{1}{Z_W(\alpha)} exp(-\alpha E_W) \tag{5.10}$$

Likewise, the error probability can be expressed as:

$$P(\omega|\beta) = \frac{1}{Z_D(\beta)} exp(-\beta E_D) \tag{5.11}$$

Finally, the distribution of posterior probability can be calculated as:

$$P(\omega|\alpha, \beta, D) = \frac{1}{Z_S(\alpha, \beta)} exp(-S(\omega)) \tag{5.12}$$

By using the Bayes' law, the optimal values of the α and β regulation parameters can be inferred from the data.

$$P(\alpha, \beta|D) = \frac{P(D|\alpha, \beta)P(\alpha, \beta)}{P(D)} \tag{5.13}$$

Where $P(\alpha, \beta)$ is the prior probability for the α and β regulation parameters, $P(D|\alpha, \beta)$ is the likelihood term, called the evidence for α and β [181]. Thereby, the minimization of $S(\omega)$ has the same results as the maximization of $P(\omega|\alpha, \beta, D)$, which depends on the α and β parameters. The optimal values of α and β are obtained from the next equation [183]:

$$\alpha = \frac{\gamma}{2E_W} \tag{5.14}$$

$$\beta = \frac{(n - \gamma)}{2E_D} \tag{5.15}$$

γ is obtained from the next equation:

$$\gamma = \sum_{j=1}^{m} m - \alpha trace^{-1}(H) \tag{5.16}$$

Where γ is the number of effective parameters, i.e. how some of the NN parameters are effectively used to reduce the error function, m is the number of parameters and H is the Hessian matrix of the objective function $S(\omega)$.

In the Bayesian context, the iterative process to find the optimization of the weights ($S(\omega)$) and the optimal values of α and β in (5.16) (γ) has to be, according to [182]:

1. Initialize the weights and the values of α and β .
2. Move a step forward in the LM algorithm to find the weights that minimize the objective function $S(\omega)$(5.8).
3. Calculate the effective number of parameters γ and new values for α and β .
4. Repeat steps number 2 and 3 until it converges.

With these parameters, the number of neurons in the hidden layer and the number of hidden layers, the model has been adjusted and the multimedia traffic classification results have been improved, as Table. 5.1 shows. All the scores have been improved by adjusting the model, except for accuracy. The recall of the model has been improved up to near a 4%. This is an important fact for the problem of this work.

Table 5.1: Optimization of the RNN classification model.

	PRECISION	RECALL	ACCURACY	F1	PROXIMITY
RNN	91.289	86.645	95.038	88.907	97.847
RNN$_{\alpha,\beta,HN}$	94.032	90.067	91.511	90.465	97.117

In conclusion, the BRNNs are more robust than the standard networks with backpropagation, and they can reduce or even eliminate the necessity of an extended cross-validation. The Bayesian regulation is a mathematic process that converts a non-linear regression into a statistical problem. The advantage of BRNN networks is that they provide solution to a series of problems such as: the model choice, the robustness of the model, the choice of the validation set, the size of the validation effort and the optimization of the network architecture. Moreover, it is hard to overtrain them, because the evidence procedures present an objective Bayesian criteria to stop the training. It is hard to overfit them too, because RNN calculates and trains in with a set of effective parameters or weights in the network, disabling those that are not relevant.

5.3.5 Classification and Modelling of Network and Video for QoE Regulation

Once the type of traffic classification model, which indicates when the traffic is critical, has been defined, some subpatterns that represent different network conditions and video characteristics are needed. In order to do that, the samples are classified according to the MOS, which is defined by the class label. Then, each subgroup is divided into clusters that are represented by a centroid or subpattern.

Extraction of the Network and Video Subpatterns

The multimedia traffic subpatterns help to recognize network conditions and possible solutions for critical traffic situations by the variation of the characteristics of the video sent. The first step to extract these multimedia traffic subpatterns was to study the correlation between the video characteristics during the transmission and the video quality perceived for the end user. This is analyzed from the results presented in Figs. 5.29-5.32. They show several network and video characteristics and their relationship with MOS. They are: frame rate, minimum frame rate, width, height, bitrate and stream size and bandwidth. Their values have been obtained from the dataset used in this work. Each value corresponds to the mean and the standard deviation of every characteristic with the same MOS value defined by the class label. As Fig. 5.29 shows, there is some connection between the characteristics of the video sent and the MOS. In Fig. 5.29 this relation can be observed with regard to the minimum frame rate. It can be observed that, with the current dataset, the bigger the MOS, the bigger the minimum frame rate. However, this trend does not occur with the frame rate, which increases with MOS until a MOS = 3 but then there are no more considerable variations. This is also presented in [74] where MOS has a high correlation with the minimum frame rate (0.4732), unlike with the frame rate (0.1528). The correlation value of Pearson exists in the interval [-1, 1]. A value close to 0 indicates a low correlation but a value close to 1 or -1 indicates a high correlation. Fig. 5.30 depicts a similar relation but declining, the biggest the MOS, the lowest the video resolution. This trend is clear and, as it is shown in [74], it coincides with a high correlation between those values (Height = -0.2902 and Width = -0.3108). As regards bitrate and stream size, this tendency is not as clear, although it is considerable (Fig. 5.31). This fact demonstrates the correlation between both measurements, and it will help to the development of the management system. The last measurement, the bandwidth, Fig. 5.32, represents a network parameter with a really low correlation of 0.1552. Therefore, this parameter will not be considered by the system as a decisive measurement for improving the critical status of the network.

After discussing the existent correlation between video and network characteristics with regard MOS, a system for extracting subpatterns will be developed. The main idea is to create clusters which can make relations between both, different network conditions and video characteristics, and the quality perceived by the end user. Therefore, for each MOS value, a set of subpatters that represent that value will be obtained. The number of subpatterns or clusters for each MOS value is unknown. In order to extract them, validation techniques will be used to select the adequate number.

The algorithm used to carry out the clustering has been K-means. The metric distance used for the calculation of the centroids is the Euclidean distance. The whole dataset X has been divided into subset $X_M \in X$, where M=1,2,3,4,5. This defines subset with regard the MOS values. Then, given an array of measurements $x_m = x_{m_1}, x_{m_2}, \ldots, x_{m_n} \in X_M$, the k sets are calculated using the K-means algorithm, minimizing the sum of squares for each set of centroids $C_M = C_{M1}, C_{M2}, \ldots, C_{Mk}$, so that:

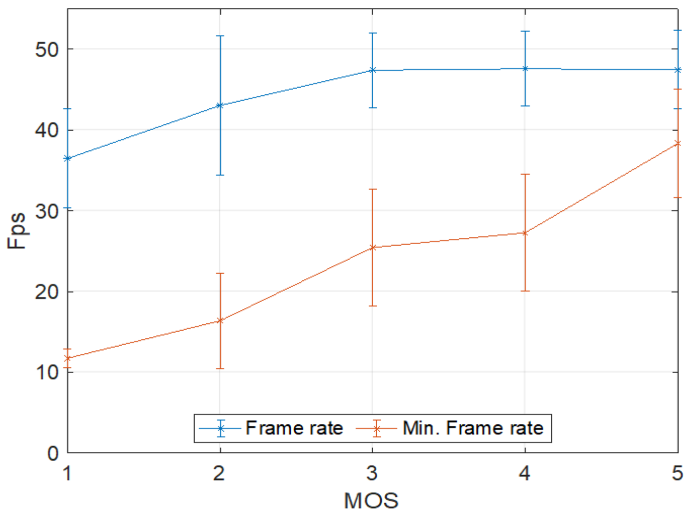


Figure 5.29: Mean and standard deviation of frame rate and minimum frame rate with regard to MOS.

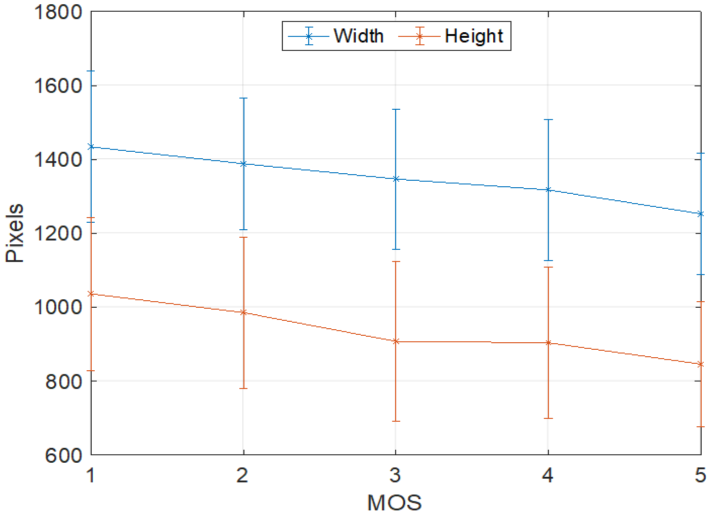


Figure 5.30: Mean and standard deviation of width and height with regard to MOS.

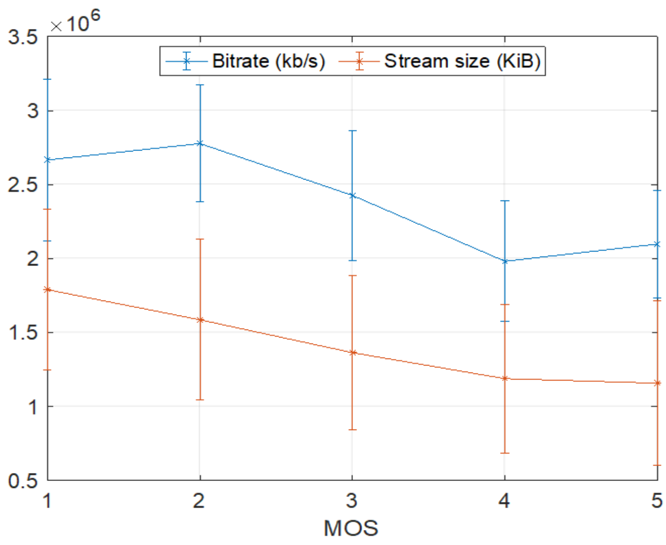


Figure 5.31: Mean and standard deviation of bitrate and stream size with regard to MOS.

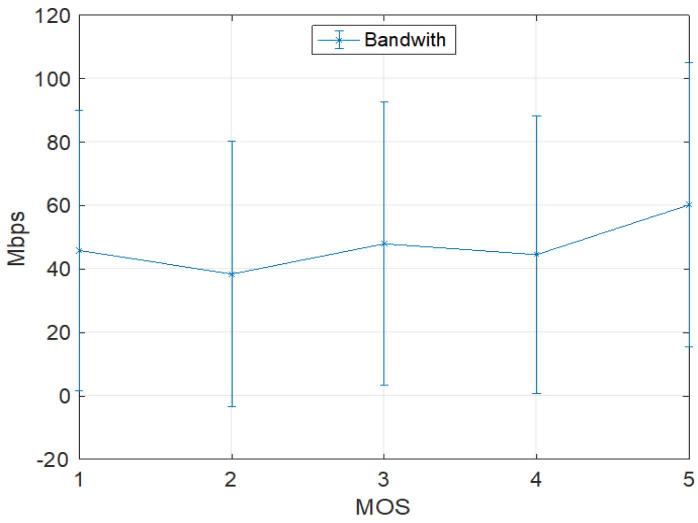


Figure 5.32: Mean and standard deviation of bandwidth with regard to MOS.

$$argmin(C_M) \sum_{i=1}^{k} \sum_{x_m \in C_{Mi}} \|x_j - \mu_j\|^2 \tag{5.17}$$

Where μ_i is the mean of points in C_{Mi} and n=14, being n the number of characteristics per sample. The result is C_{Mk} centroids for X_M data sets.

Once the algorithm used for the clustering has been defined, the optimal number of clusters will be calculated. The validation method used to define the number of clusters is Silhouette. The statistic of Silhoutte works with distances. This method traces the silhouettes using the function of distance between points specified in the distance metric, which in this case is the Euclidian distance. In Fig. 5.33 and Fig. 5.34, the results obtained for the subsets defined for MOS = 4 and MOS = 5 after applying the Silhoutte method are depicted. The optimal number of clusters is 4 for the subset of data defined by MOS = 4 and 5 for the one defined by MOS = 5. To begin with, negative values represent errors in the selection of a cluster. However, there are no negative values in Fig. 5.33 nor in Fig. 5.34. That means there are no values assigned to a cluster where they actually do not belong. Another relevant fact, which determines the validation of the clustering, is that the values obtained must be higher than 0.8. In both graphs, the majority of values are higher than 0.8. That means that most of the area under the value of maximum probability equal to 1 is covered. Once defined the number of clusters and once the centroids have been extracted, the working of the system is defined in the next subsection.

Subpattern-based QoE Regulation Model

Once the subpatterns are defined, the system has to be able to, given a set of network parameters, obtain video characteristics, or vice versa, to solve the situations where the traffic is critical. In order to achieve this, the next process is performed.

Firstly, the minimum Euclidean distance between the network parameters of the input and every subpatterns or centroids is calculated. The network Euclidean distance between the input sample $x = x_1, x_2, \ldots, x_n$ and the centroid $c = c_1, c_2, \ldots, c_n$ is:

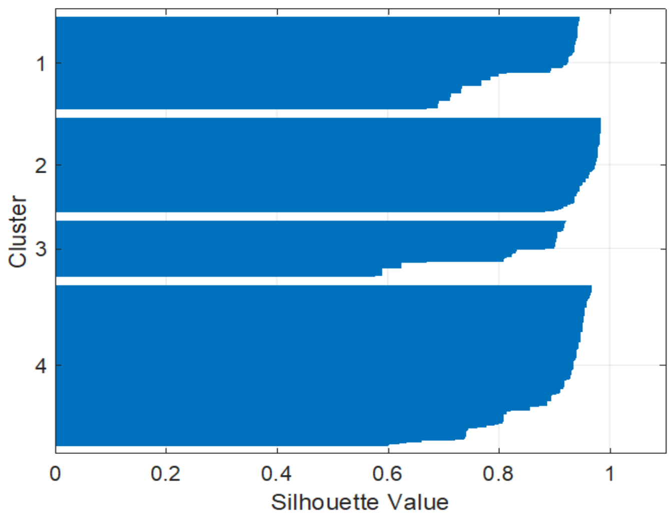


Figure 5.33: Subpatterrns for MOS = 4.

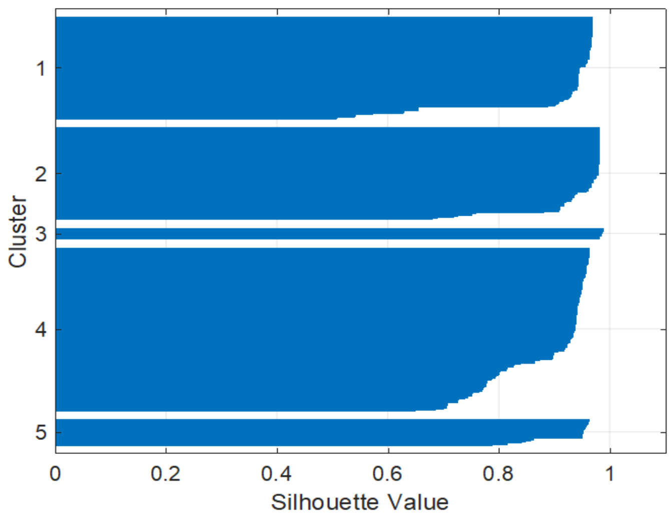


Figure 5.34: Subpatterrns for MOS = 5.

$$\begin{aligned}
d((x_1, x_2, \ldots, x_r), (c_1, c_2, \ldots, c_r)) = & \\
&= d((c_1, c_2, \ldots, c_r), (x_1, x_2, \ldots, x_r)) \\
&= \sqrt{(c_1 - x_1)^2 + (c_2 - x_2)^2 + ... + (c_r - x_r)^2} \\
&= \sqrt{\sum_{i=1}^{r} (c_i - x_i)^2}
\end{aligned} \tag{5.18}$$

Where r is the number of network parameters for each sample and whose value is 5. The set of centroids that define a non-critical traffic, i.e. with MOS = 5 and MOS = 4, is defined as:

$C = (c_{t,1}, c_{t,2}, \ldots, c_{t,n}) | C = C_4 \cup C_5 \wedge t = K_4 + K_5$,where K_M is the total number of centroids of C_M.

Therefore, the minimum network distance to each subpattern is defined as:

$$argmin(c_{j,i}) \sqrt{\sum_{i=1}^{r} (c_{j,i} - x_i)^2}, \quad j \in [1, t] \tag{5.19}$$

(5.18) returns the subpattern or centroid j whose network conditions are similar to the input sample. From this result, the video characteristics of that subpattern are extracted. Then, the video characteristics will change until the critical network situation will be solved. The Pearson's lineal correlation coefficient between the objective QoE and each characteristic determines the probability that the system chooses one characteristic or other. The higher the probability of a characteristic is, the higher priority of changing that characteristic has. The probability function of $c_{j,i}$ is defined as:

$$f(c_{j,i}) = f(x_i) = \frac{\sum_{s=1}^{W} (x_{s,i} - \overline{x})(y_s - \overline{y})}{\sqrt{\sum_{s=1}^{W} (x_s, i - \overline{x})^2} \sqrt{\sum_{s=1}^{W} (y_s - \overline{y})^2}} \tag{5.20}$$

Where W is the total number of samples of the dataset X, Y is the set of class label for each sample and i the characteristic index.

The probability of selecting the characteristic i of the centroid j from the centroids set C is defined by $P(c_{j,i})$. The order of the index depends on the maximum probability value that maximizes the value of the function that is defined as:

$$i = argmax(i \in I)f(x_i, i) \tag{5.21}$$

Where $I = i_5, i_6, ..., i_4$ is the set of indices to be estimated, which correspond to the video characteristic indices. If the estimated index did not solve the network critical state problem, the next index with the maximum probability would be searched.

5.4 Robust Multimedia Traffic Management System

After configuring the multimedia traffic classification model and the QoE regulation model, a robust multimedia traffic management has been developed. The system analyzes and classifies the multimedia traffic, detects if it is critical and, if so, finds the video parameter that, by being changed, can solve the problem. The system is considered as robust because it learns from the cases that are not solved by applying the method discussed in last subsection, creating new subpatterns. If the problem is not solved, the network conditions and the video characteristics are saved to feedback the system. In this section, the architecture of the network is presented and the interaction between the modules defined is discussed. Then, the algorithm and its implementation are detailed. Finally, the test bench is discussed.

5.4.1 Network Architecture

The classification model, as a system, can be applied in every kind of network because the performance of the system does not depend on the kind of network, but only on the network parameters (jitter, delay, packet loss and bandwidth) monitored on the nodes." However, in this work, we use SDN as a tool to provide the system with data. Consequently, we extract statistics directly from the network nodes. Despite the independency of the system towards the network architecture, our proposed network presents the architecture displayed on Fig. 5.35, as an example. The blue lines are the corresponding ones to the datapath, which is used to communicate the data through the network nodes. However, the connections painted in green belong to the SDN Control to Data-Plane Interface (CDPI), which physically implements the interaction between network nodes and the SDN controller. The final nodes are IoT nodes, which are the producers of video traffic. However, the core of the network, starting from the gateway, is made of nodes of an SDN network. Therefore, there is an SDN controller which manages the network. SDN controller receives the statistics gathered by the network nodes. These statistics are network parameters: delay, jitter, loss rate and bandwidth. From these parameters, the management system determines if a great QoE degradation can happen. As in the rest of the dissertation, OpenFlow protocol is used to implement the communication between the nodes and the SDN controller. Furthermore, these statistics are needed by the system to perform several tasks. For instance, classifying the critical traffic is one of the tasks that needs the data from the network in real time. In order to explain this system, it is supposed that the AI module is implemented through the northbound API. Consequently, we define the global logical architecture of the network as is depicted in Fig. 5.36. It depicts how the communication between modules is implemented. Firstly, the communication between the SDN controller and the network nodes is performed through the southbound API. The northbound API is the channel between the SDN controller and the AI module. In this interaction, not only the network statistics and data are sent to the AI module. The AI module sends back the necessary actions to be performed in the network to ensure the QoS. The video streaming application receives orders from the SDN controller. This allows the SDN controller to execute the orders of the AI module. Depending on the results of the system, the SDN controller will change the video characteristics. This is performed by using the interface of the video streaming application. The software features are presented in Subsection 5.4.4.

5.4.2 Simple Management Algorithm

Fig. 5.37 shows the algorithm that describes in detail how the management system works. Every 2s, which corresponds to a GOP, the system gets as input the QoS and video parameters. According to those parameters, the classifying model estimates if the traffic is critical or if it is not. If it is not critical, the previous process will be repeated. However, if the traffic is critical, the process to solve this critical situation will start. The analyzed traffic subpatterns are those which can recover a MOS higher than 3. That means, they are te ones that can exit from the critical situation. Those subpatterns are obtained from the clustering, explained in Subsection 5.3.5, applied to the multimedia traffic patterns.

When the subpatterns have been obtained, the system calculates which parameters of the transmitted video can be modified to improve the QoE. The

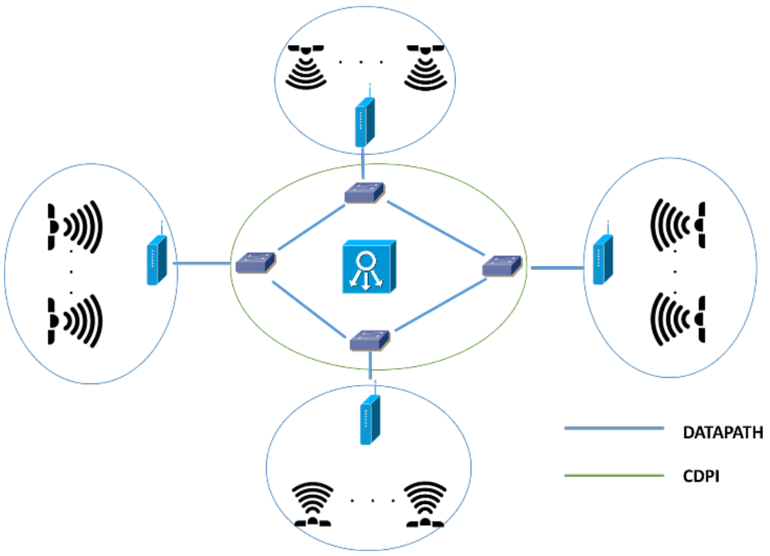


Figure 5.35: Network architecture example.

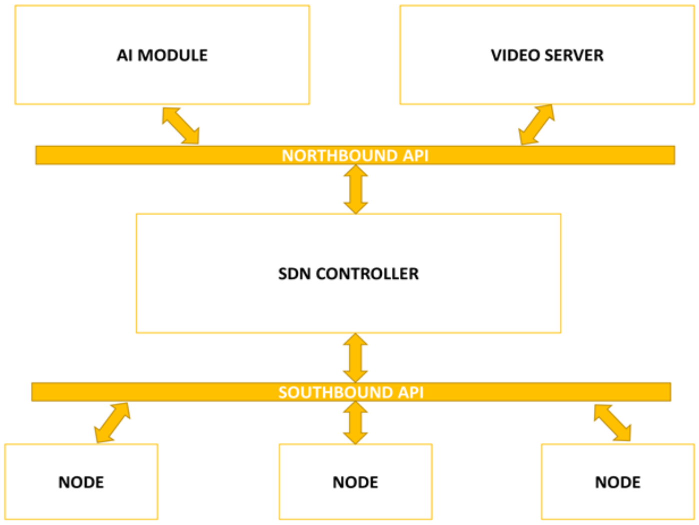


Figure 5.36: SDN Architecture used in this system explanation.

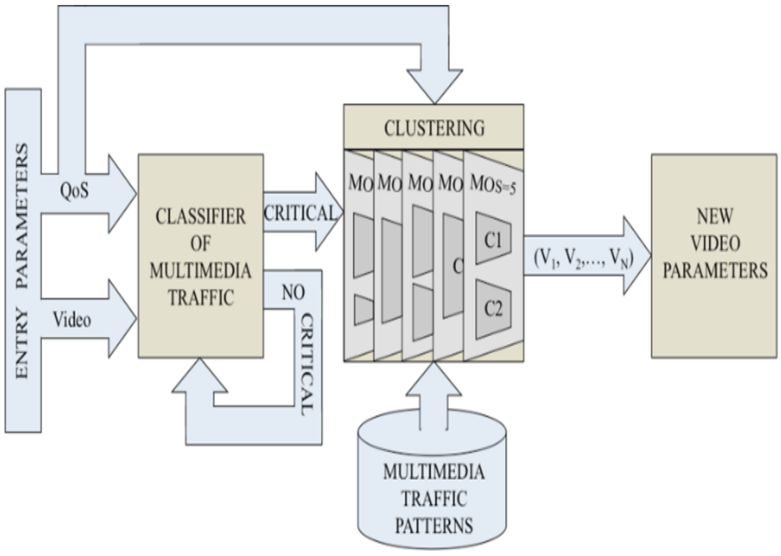

ENTRY PARAMETERS
QoS
Video
CLASSIFIER OF MULTIMEDIA TRAFFIC
CRITICAL
NO CRITICAL
CLUSTERING
MOS=5
C1
C2
($V_1, V_2, \ldots, V_N$)
NEW VIDEO PARAMETERS
MULTIMEDIA TRAFFIC PATTERNS

Figure 5.37: Management system algorithm.

system uses as input parameters the QoS parameters and the different traffic subpatterns extracted from the clustering. Then, the shortest distance to every subpattern according to the network parameters is calculated. Thereby, the system looks for the subpattern, or centroid, that represents the closest or most similar network conditions to the current ones. Once the subpattern has been obtained, the video characteristics of that subpattern are used as the new video transmission parameters. The selection of the video characteristic is performed regarding to the model explained in the last subsection. Specifically, we will use (5.21). If the critical traffic situation is not solved, the next characteristic, according to the model, is chosen. If applying this process the traffic critical situation is not solved, then we would apply the robust method of management. In this way, instead of changing only one different characteristic, each of them will be added or combined to the transmitted video. If this procedure does not work, we would keep this sample in order to improve the system with new intelligent methods that we will study in future works. This process is repeated until the problem is solved or there are no more characteristics to choose. As the subpatterns are defined by both, network parameters and video characteristics, any of them could be the input parameter or the parameter to be adjusted. This feature of the system will be discussed further in future works.

5.4.3 Robust Management System Implementation

In this subsection, the implementation of the robust management system is described. Algorithm 5.1 details the main process, while Algorithm 5.1.1 details the steps performed after the main loop. For each input cycle (GOP of 2s), the program reads the average network and video parameters, and an array of indices is initialized. A loop starts until the array is not empty. In this loop, the system checks the multimedia traffic using the BRNN, trained and validated in the learning process. If the traffic is not critical, the array is cleared and the program restarts. Nevertheless, if the traffic is critical, firstly, the index of the subpattern with the lowest Euclidean distance, by using (5.19), between the network parameters of the subpatterns matrix (SUBPATTERNS) and the network parameters of the input sample (X). Then, the new video characteristics (NEWVIDEOparam) are obtained from the subpatterns matrix of that index. After that, the index of the characteristic with the highest correlation from NEWVIDEOparam is calculated, by using equation (5.21), and, then, it is removed from the array of indices. Finally, this characteristic is changed in the video that is being transmitted. If, after changing the characteristic, the traffic becomes non-critical, the program will leave the while loop, clearing the array of indices I and it will restart. If the critical situation is not solved, the next characteristic with the highest correlation will be selected. This is repeated until the program finds a characteristic that solves the critical situation or until the array of indices gets empty. If the problem is not solved, a robust solution is taken. This solution consists of combining different characteristics attending to (5.21) until a solution is found. This process is performed if there has not been any other similar case performed with this robust method. The robust solution is determined by the Euclidean distance, using (5.19). If there is no other similar case and the system finds a solution, it calculates the distance to the centroids of the clusters and a new subpattern, along with its solution, it is saved (ROBSUBPATTERNS). If the system does not find a solution, the case is saved as a sample to study to improve the system (UNKNOWN). The intelligent system to improve it with this feedback is not studied in this work, but it can be discussed in future works. In this last scenario, the program saves the sample so that the network can learn from it (UNKNOWN).

Algorithm 5.1 Robust Management System Algorithm

```
1:  X1 = QoSparam
2:  X2 = VIDEOparam
3:  I = ALL_INDEX
4:  indexsubpat = min_dist(SUBPATTERNS(4:5), X1)
5:  NEWVIDEOparam = SUBPATTERNS(indexsubpa , 6:14)
6:  While size(I) not NULL and SOLVED = false
7:      TRAFFIC = net(X1 , X2)
8:      If TRAFFIC = CRITIC
9:          If NOT ROBUSTMETHOD
10:             indexcarac = maxcorr(NEWVIDEOparam, I)
11:             X2(indexcarac) = NEWVIDEOparam(indexcarac)
12:         Else
13              ROB_INDEXScarac =
                 robust_maxcorr (NEWVIDEOparam, I)
14:             X2(ROB_INDEXScarac) =
                 NEWVIDEOparam(ROB_INDEXScarac)
15:         End If
16:         I = delatecarac(I, indexcarac)
17:         If size(I) is NULL
18:             TRAFFIC = net(X1 , X2)
19:             If TRAFFIC not CRITIC
20:                 Go to 6
21:             End If
22:         End If
23:     Else
24:         SOLVED = true
25:         If ROBMETHOD
26:             ROBSUBPATTERNS(f++)=NEW ROBSUBPATTERN
27:         End If
28:     End If
29: End While
```

Algorithm 5.1.1 Robust Management System Algorithm (Cont)

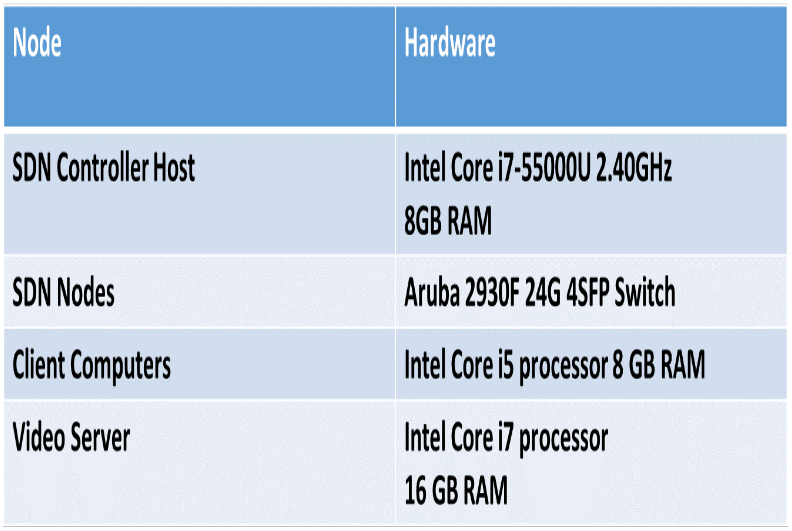
```
30: If SOLVED = false
31:     If NOT ROBMETHOD
32:         indexsubpat_robust =
             min_dist(ROBSUBPATTERNS(:,1:5), X1)
33:         If indexsubpat_robust = NULL
34:             ROBMETHOD = true; I = ALL_INDEX;
35:             X2 = VIDEOparam
36:             Go to 6
37:         Else
38:             NEWVIDEOparam =
                 ROBSUBPATTERNS (indexsubpa , 6:14)
39:             X2 = NEWVIDEOparam
40:         End If
41:     Else If SOLVED = false and ROBMETHOD
42:         UNKNOWN(fk++) = [X1 X2]
43:     End If
44: End If
```

5.4.4 Test Bench

In this subsection, both the hardware and the software used are specified. In Fig. 5.38, the hardware of the SDN nodes is shown.

Regarding the software, the SDN controller used in these tests is the one presented in chapter 3. It has been created from scratch (programming by us, not using one of the available SDN controllers like OpenDayLight) so that we could perform customized researches such as [184]. This allows a better control over the operation and guarantees the inter-operability between the software modules described by the network architecture. As video server software, VLC has been chosen due to its capability to modify the video streaming dynamically. Therefore, the SDN controller can modify the video characteristics. The version used is VLC 3.0.7.1 for both, the video server and the client computers.

In the next subsection, the results of the test are discussed.



Node	Hardware
SDN Controller Host	Intel Core i7-55000U 2.40GHz 8GB RAM
SDN Nodes	Aruba 2930F 24G 4SFP Switch
Client Computers	Intel Core i5 processor 8 GB RAM
Video Server	Intel Core i7 processor 16 GB RAM

Figure 5.38: Hardware used in the test bench.

Working Test

The last study performed in this section is a working test of the developed system. The test samples are provided to the system and the system check for each sample if the traffic is critical or it is not. If the traffic is critical, the system tries to solve this situation by changing the video parameters as it has been described in this section. Two different experiments have been performed, in the first one all the centroids or subpatterns are used and in the other one some of them were removed to analyze the consequences. In order to improve the robustness of the system, the network and video parameters are saved to feedback the system, only if the problem has not been solved.

1) First Test

In the first experiment, the performance of the system when all the multimedia subpatterns are used is evaluated. If the system detects a multimedia traffic critical situation, the network status at that moment will be the input parameters. Then, with those parameters, the system will search for the subpattern that matches the conditions given. The Euclidean distance is the method which is used to make this comparison. The lowest distance determines the subpattern to be selected. Once the subpattern has been selected, the video characteristics will be modified to solve the critical situation of the network.

In this first experiment, every video characteristic was checked to observe if the critical situation changed. Fig. 5.39 shows the results of the test. If the system is allowed to select any centroid, it has a 100% success rate.

As it is shown in the figure, the characteristics that more critical traffic situations solve are: Frame count and height, with about 60%. Then, with about 50% stream size, width and min. frame rate. The characteristic with the worst results to solve critical traffic situations is the frame rate, with only a 15% of all the cases.

A statistical study has been performed from the obtained results. The data presented in Fig. 5.40 shows that the system uses bitrate as characteristic to be changed when the delay is high (over 0.03s), frame rate and stream size with delays values over 0.0073s and frame count, proportion stream, footer size, width and minimum frame rate when the delay is low (0.003-0.0012s). The results as regards jitter are similar. However, the differences between low and very low jitter values are not as big as with regard to delay. These values are 0.004s for a low jitter and 0.002s for a very low jitter.

The loss rate can be divided into three categories: high loss rate (4%), medium loss rate (0.5%-0.2%) and low loss rate (0.08%). In order to solve a critical situation with a high loss rate, the characteristics used are the stream size, the bitrate and the frame rate. These characteristics are related to the video bandwidth consumption. In scenarios with network conditions with loss rate of 0.2%-0.5%, the video characteristics changed are: frame count, footer size and height. Finally, when the loss rate is lower than 0.008% these parameters are: proportion stream, width and min. frame rate.

2) Second Test Like in the last scenario, a statistical study has been performed from the obtained results. Nevertheless, if the system can only select a specific number of centroids and the others are discarded, the problem is not always solved. This happened in two cases. In order to solve those two cases, the robust method of modifying two different characteristics together, depending on the subpattern, was used in this study. Fig. 5.41 shows the test result for every video characteristic checked to observe if the critical situation changed. As the figure shows, and unlike the last case (Fig. 5.39), the characteristic that solves more critical traffic situations is the footer size with about 55% of cases. The next characteristic is the minimum frame rate, with about a 50%. Stream size, bitrate and height are able to solve about a 40% of critical traffic situations. The characteristic with the worst results, like in the last case (Fig. 5.39), is the frame rate with less than the 15% of the cases. It can be observed that the frame count in the last case and the footer size in this

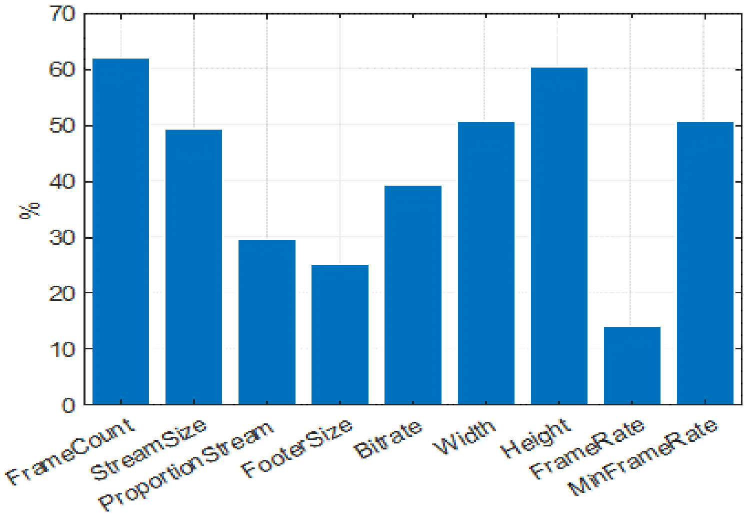


Figure 5.39: Video characteristic selection performed by the system according to the NN model (based on QoE estimation) to solve the critical situation with all the subpatterns.

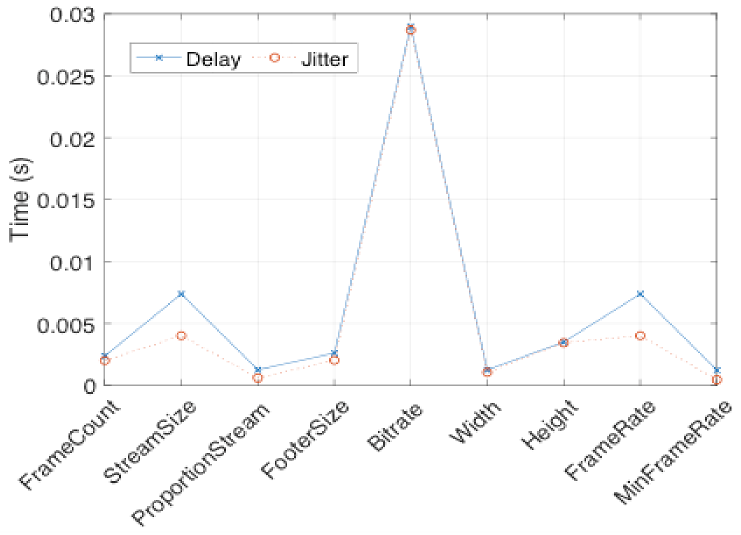


Figure 5.40: Results of the statistical study relationship jitter and delay with every video characteristic selection.

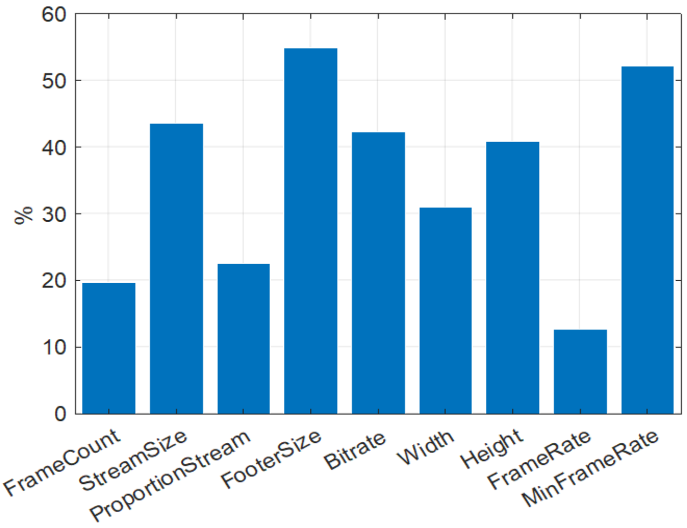


Figure 5.41: Video characteristic selection performed by the system according to the original model (subjective QoE) to solve the critical situation without all the subpatterns.

one, solve an important quantity of cases, despite being characteristics with low correlation. This will be an interesting aspect to be addressed in future works regarding the improvement of the characteristics selection by using the management system. As it is depicted in Fig. 5.42 when there is a high delay, the parameters changed are the bitrate and the frame rate of the video. When there is a low delay, the other parameters are used. The delay is considered high when it is higher than 0.03s and low when it is lower than 0.007s. When the delay is around 0.07s, the parameter changed is the stream size. The frame count, footer size and height are variated when the delay is over 0.002s. Finally, when the delay is 0.0012s, the parameters used are proportion stream, width and minimum frame rate. The behavior regarding jitter is similar, although, for the stream size parameter, the value is decreased to 0.004s from the 0.007s with regard to delay. The loss rates can be classified into high loss rate (4%), significant loss rate (1%) and low loss rate (0.2%-0.8%). In order to solve a critical situation with loss rates greater than 4%, the parameters changed are bitrate and frame rate. The stream size is used with losses of 1% and the other parameters for losses of 0.08-0.2%.

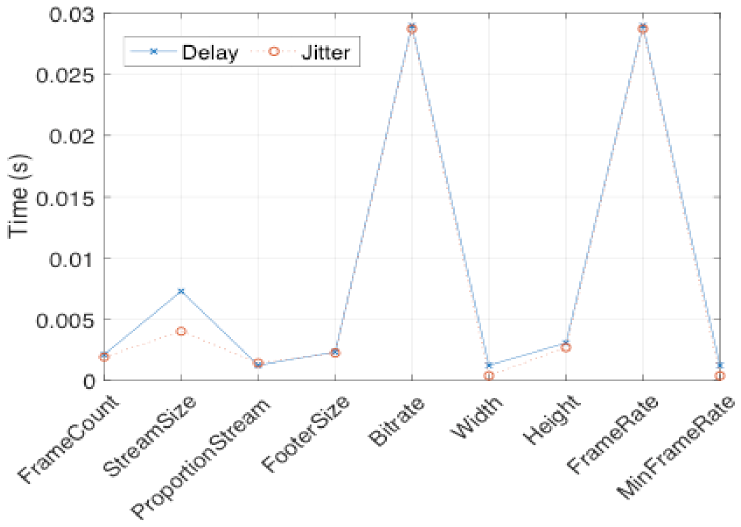


Figure 5.42: Results of the statistical study relationship jitter and delay on the second test with every video characteristic selection.

5.4.5 Conclusion

In this subsection we have developed a multimedia traffic management system to classify the type of traffic and, in case of critical situation. This tool is able to reverse the situation by changing the parameters of the video that is being transmitted. For the development of this system, first we have analyzed different classification and regression models for the QoE estimation. The models that better fit the proposed estimation problem are the classification CSVM and the regression RNN. We have performed a test to analyze them in more detail and we have concluded that, although both of them have very similar precision values, RNN adjusts better to the problem. The expected result is very close to the one we get with this method. In the following study we have analyzed and evaluated the same models from the previous study case to check how well they fit the problem of multimedia traffic classification. Again, CSVM and RNN were the models providing the best results. RGP, CDT and CKNN are also good models. They provided fair results in both studies but not so good as CSVM and RNN. Based on the results obtained, we decided that the model based on RNN fits the problem better. Next step was to define the model and the architecture based on BRNN. We also made some adjustments in order to get better statistical classification results, improving the results in the measure of Recall by 4%. Thus, after a deep study and development of a QoE estimation system, we have been able to improve the results of multimedia traffic classification obtained in the study presented in [74].

The system consists of a traffic classification module and a module to reverse critical situations of the network. Once detected a critical state of the network, this module looks for network conditions that are similar to different conditions classified in subpatterns. When the subpattern is found, it will give us a series of video features that will be applied to the video that is being transmitted, in order to reverse the critical traffic situation. The selection of these characteristics is based on the maximum correlation with respect to the objective QoE.

Once we have selected the multimedia traffic classification model and the QoE regulation model based on subpatterns, we implement the management system. The system works in the following way: first it captures samples every 2s (based on GOPs), then it establishes if it corresponds to a network critical situation. If a critical situation occurs, the system checks which subpattern has more similar network conditions. This subpattern will give us the characteristics that we should apply to the video that is being transmitted until the critical situation of the network reverts. A system implementation, the network architecture, using the SDN technology, and the proposed algorithm were discussed. After the tests we made with the system, we have verified that minimum frame rate, resolution and stream size are highly efficient when the system is reverting a critical situation. We also have verified that with all subpatterns, all critical network situations have been solved. If we eliminate subpatterns, this fact would not occur.

5.5 A Proposed Machine Learning System to Solve Multimedia Transmission Problems

In this section, the system that choose the best action to perform, which is the last task of the AI module and the routing protocol. First, a brief reminder of the architecture is introduced. Then, some network considerations are addressed. After that, the machine learning algorithm is discussed in detail. After that, a comparison with other similar approaches is provided. Finally, the results of the test performed to the system are presented and the reward updated evaluation problem is addressed.

5.5.1 Architecture

The scenario used in this proposal is the same as the one used in chapter 4. The SDN network is entirely managed by a single controller. This controller must not only decide the path each packet has to follow but also has to take actions to prevent transmission problems. In order to achieve this, the controller is composed by several modules. These modules are shown in Fig. 5.43. The modules viewed in chapter 4have been expanded. Now, the AI module is split into the statistic analyzer (the QoS degradation estimator explained in the last section) and the action chooser (the system described in the current section). Likely, the SDN controller is composed by two different modules: the routing module and the communication module.

As Fig. 5.43 shows, the communication module provides the message exchanging between the controller and the network nodes, i.e., the switches. The communication is entirely based on the Openflow standard. Therefore, the network nodes must be Openflow-enabled devices. By using Openflow messages, the controller gathers data about the capabilities of the Openflow-based network nodes. Then, this communication with Openflow messages allows the controller to generate orders with the actions the nodes should perform. The routing module is in charge of detecting the network topology and creating paths. Moreover, it analyzes the nodes characteristics. All the details of this module have been already explained in Section 4.3.

The third module on the SDN controller is the Statistic Analyzer which analyzes the statistics sent by the network nodes to the SDN controller. These statistics are provided by the communication module. The analysis is performed in order to detect the different problems during the multimedia transmission. Furthermore, it is the module that notifies changes in the transmission status. These states are defined in Subsection 5.5.3.

The last module is the Action Chooser which uses reinforcement learning to determine the action the SDN controller should take in order to solve the current problem that the Statistic Analyzer has detected. It receives the route characteristic from the routing module in order to know which actions can be performed in which paths. In addition, the Statistic Analyzer notifies changes in the state of the transmission and its QoS degradation. With this data, the Action Chooser calculates the reward of the last action and choose another action if required. Subsection 5.5.3 describes the reinforcement learning algorithm and the concepts whose definition is adapted to the problem.

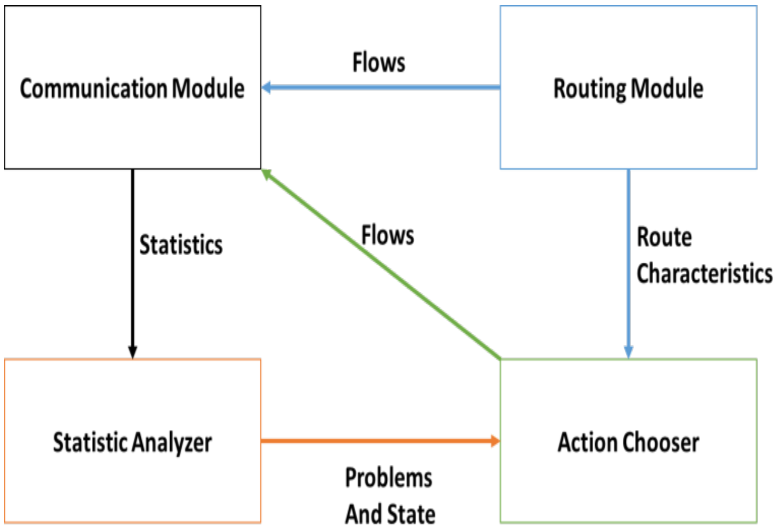


Figure 5.43: SDN Controller Modules.

5.5.2 Network Considerations

This subsection explains several concepts that the AI module handles. These concepts are needed to correctly evaluate the state of the system. Fig. 5.44 shows an example of the network we use to test our proposal.

As we can see, the scenario is composed by a multimedia server and several multimedia clients that will perform the multimedia communication. Although there are proposals that allow multicast flows on SDN [185], in this work, the multicast transmission will be treated as N unicast flows, to make easier to explain multimedia management. This assumption can be done because the system is independent to the unicast or multicast property of the multimedia transmission. In addition, to reach a destination, the flow will go through several nodes from the source to the destination. This route will be composed by two different links. On the one hand, outside links are the links between the clients and the first network node. On the other hand, inner links are the ones that interconnect two or more network nodes. This differentiation will be useful for the AI module in order to determine which action should be taken. The actions may vary depending on the characteristics of the link. If the link is an inner link, some actions like changing the route would be able to improve the transmission quality. However, if it is an outside link, some other actions like reducing the throughput of the other host could fit better. In addition,

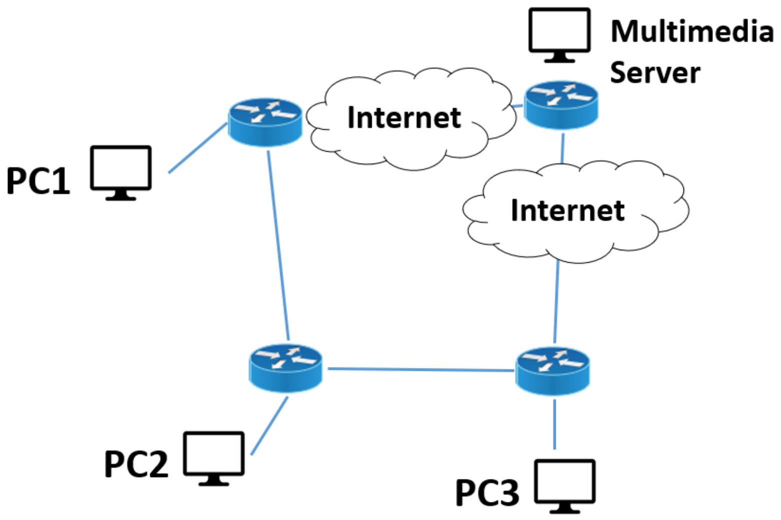


Figure 5.44: Network Example.

some links are backup lines that are not used unless they are required to ensure the QoS requirements of the multimedia transmission.

Nevertheless, the differentiation between links is not the only one done in the system. Depending on the network topology, nodes can also be used as backup nodes or as load balancing nodes. Both of them are nodes that get to the same intermediate node between the source and the destination. The difference between them is that backup nodes are only used when they are needed. However, the load balancing nodes are nodes that allow the controller to send the packets to an alternative route. The identification of the alternatives routes or backup lines/nodes is the task of the Routing Module and it is transmitted to the AI module, as it is explained in chapter 4.

5.5.3 Reinforcement Learning Algorithm

This subsection describes the reinforcement learning algorithm which is defined in two steps. First, the environment of the system is defined, and the states discussed. The system needs some data structures to operate correctly. These data structures are explained then. After that, the concepts about rewards, policy and objective are defined. Later, the actions are described, and it is explained how they are performed with the Openflow standard. Finally, the algorithm is depicted.

Environment and States

In this subsection, every single element that the algorithm uses is explained.

Firstly, we have to understand that the network is composed by elements with different roles. The most important one is the element that acts as an agent. The agent of our algorithm is the SDN controller. It is the element which will perform the actions and that will analyze the current state of the system. The network nodes are not agents and they will only modify their OpenFlow tables in order to execute the commands of the SDN controller. So, they play a less important role.

The environment of the algorithm is composed by the network and the current state of the network based on the statistics. So, the agent, i.e., the SDN controller, modifies the environment through its actions, although it will actually modify the state of the network. That means that the statistics they gather from the nodes will measure the effects of the performed actions. These actions will change the transmission performance in terms of QoS parameters.

The state of the network is related to the problems suffered by the multimedia transmission. Some of the typical problems are the lack of available bandwidth, a high loss rate, too high delay, too high jitter, etc. The SDN controller receives an alert from the Statistic Analyzer and determines the best action to perform depending on the current state. The current state can be represented as a state-machine (see Fig. 5.45). The default state for every multimedia flow is the “No Problem” state. Along the transmission of the multimedia stream, several problems may occur, and the network will be in charge of adapting its functioning to the new situations. The problems and states defined in this paper are mainly related to QoS parameters. The bandwidth, delay, jitter and loss rate are the parameters the Statistic Analyzer monitors. Besides providing the Action Choose with the measures of the problems, it evaluates these parameters on the multimedia flows in order to notify the state changes. When a problem is detected, this leads the system to a new state related to that problem and the system will remain in that state until the problem is solved. Consequently, the system will return to the “No Problem” state. However, some of these problems could be combined and the current state could even change to a worse state. This can happen when the selected action did not generate the required effect, or it was performed too late. Fig. 5.45 shows all these states and changes between them. All those states allow coming back to a better state when the problem is solved (or its QoS parameter is improved).

It is expected that the actions performed change the states permitting the system remains on states with good performance. The best scenario would be remaining in the "No Problem" state. However, in an actual scenario the problems happen and so, it will better to be in states with one problem than being in states with combined problems. In order to achieve this, the following actions can be chosen by the SDN controller:

- Wait
- Switch to a backup line
- Switch to a backup node/path
- Load balancing
- Use QoS-Guaranteed Queues
- Tag packets with VLAN PCP
- Limit the available bandwidth for a flow

This set of actions is only a proposal. More actions can be added to the set. The requirement is that the routing module must be able to perform the actions. Some other recommended actions (based on the QoE studies previously presented) are:

- Change Framerate of the video
- Change Bitrate of the video

These actions cause changes in the network performance, and, depending on the next values obtained from the Statistic Analyzer, the reward value of the action taken will be updated.

The way the rewards are assigned and calculated and how the actions are performed are described in the next subsections. However, we need to firstly know the data structures and concepts.

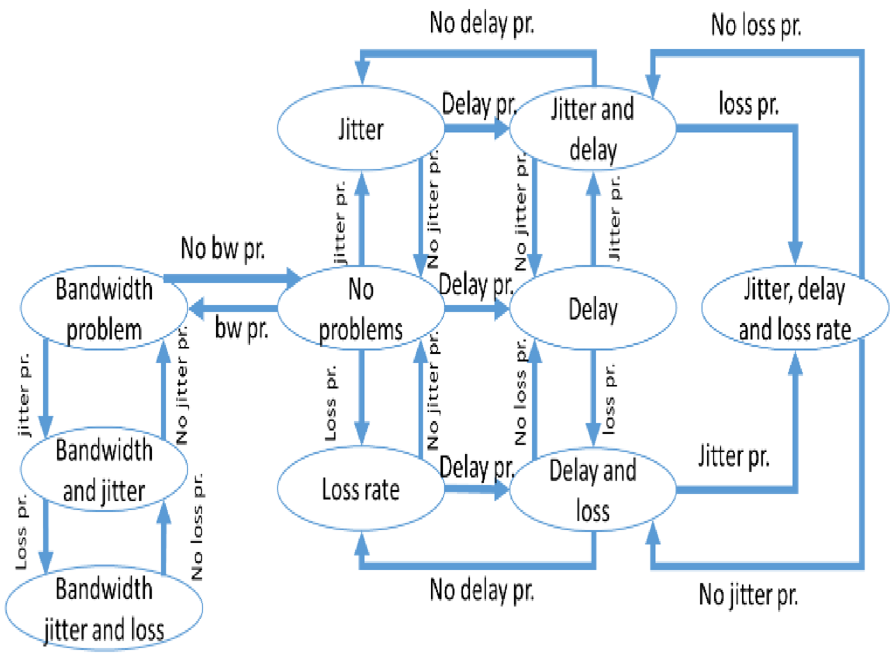
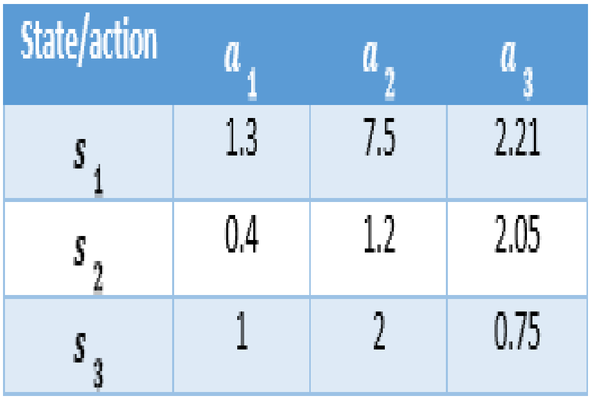


Figure 5.45: Diagram of the states of a multimedia flow in the system.



State/action	a_1	a_2	a_3
s_1	1.3	7.5	2.21
s_2	0.4	1.2	2.05
s_3	1	2	0.75

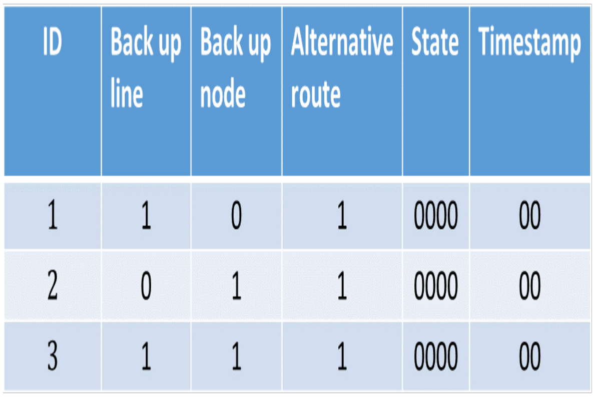
Figure 5.46: Example of Array H.

Data Structures

In this subsection, the data structures that the algorithm needs are described.

The algorithm needs two different data structures in order to work. First, a two-dimensional array it is required to store the rewards of each action-state pair. This array, called H, contains the reward obtained for each combination of action and state. The structure of H is shown in Fig. 5.46. This structure is independent of the number of multimedia flows. The values shown in the table are examples. Usually, the algorithm should begin with the same value for every entry. However, specific initial values could fit to some scenarios.

Nevertheless, the system should independently manage each multimedia flow. That means the actions are caused by changes in the state of a single multimedia flow. Keeping the state of each multimedia flow, and its route characteristics, it is important to get the highest performance. Therefore, the Multimedia Management array (M) is defined. The array M (shown in Fig. 5.47) is a two-dimensional array that contains the status of each multimedia flow. Moreover, it is also composed by a set of boolean flags which indicate if the related actions can be performed for that flow or not. Furthermore, the current timestamp is also stored in this array. In Fig. 5.46, the fields are shown in bits in order to easily see the size needed to store this data All the data related to the capabilities of the route are provided by the routing module.



ID	Back up line	Back up node	Alternative route	State	Timestamp
1	1	0	1	0000	00
2	0	1	1	0000	00
3	1	1	1	0000	00

Figure 5.47: Example of Array M.

Rewards, Policy and Objective Function

The reward obtained due to taking one action or another depends on the problem itself and on the metrics that show how good or bad the performance of our system is. Typically, reinforcement learning is applied to games, especially video games. In those games, the reward obtained depends on the score obtained in that game. Actions like earning a coin or defeating an enemy produce a big reward, because they increase the score. In our system, the goal is to avoid the multimedia transmission problems. Therefore, the metrics used are the QoS measurements. Consequently, the rewards are assigned depending whether the problems are being reduced or not.

First, it is important to highlight that in order to adapt reinforcement learning to the multimedia transmission in SDN, the rewards must be given attending to the effects that previous actions have produced. That means, that the effects are not suddenly perceived, so the reward from an action must be calculated in several timestamps or iterations. So, the greater effect will not be perceived in the first timestamps. This way of operating, with several intervals, introduces two new definitions, i.e., the time between iterations (τ_{it}) and the number of iterations (n_{it}) before calculating the new reward. Then the time (τ_{rew}) the system needs to take an action and to calculate the reward is defined in (5.22).

$$\tau_{rew} = n_{it} * \tau_{it} \tag{5.22}$$

Consequently, the reward is an array of (n_{it}) elements, one for each timestamp measured. As an example, taking $n_{it} = 3$ the reward will be defined as (5.23).

$$r = [r(1), r(2), r(3)] \tag{5.23}$$

Each element r(t) of the array is calculated as a difference in terms of the multimedia transmission problems of the two last timestamps, p(t-1) and p(t) (5.24).

$$r(t) = \rho(t-1) - \rho(t) \tag{5.24}$$

, where $\rho(t)$ is the measurement of multimedia transmission problems at timestamp t.

This calculation is done taking into account QoS parameters like delay, jitter and loss rate. In addition, the result is 0 if the difference between both terms is negative. This measurement is calculated and returned by the Statistic Analyzer module.

In order to take the total reward R of an action, the individual rewards for each timestamp r(t) must be multiplied by a weight ($t/2^{t-1}$). Due to the fact that for our system is more important the effect in medium and long term that the immediate response, the total reward can be expressed by (5.25).

$$R = \sum_{t=0}^{n_{it}} r(t) * \frac{t}{2^{t-1}} \tag{5.25}$$

The weight applied to the individual reward value is identified by γ (5.26).

$$\gamma = \frac{2^{t-1}}{t} \tag{5.26}$$

Fig. 5.48 shows the evolution of the value of γ as a function of the timestamps. As we can see, these values start to increase after the second timestamp. That means the action will be more rewarded if there is a medium-term improvement rather than only a short-term improvement. This γ factor has been chosen to manage the first 4 timestamps. If a modification of the system is required or wanted, the γ should be adjusted to the timestamp frame. For example, in

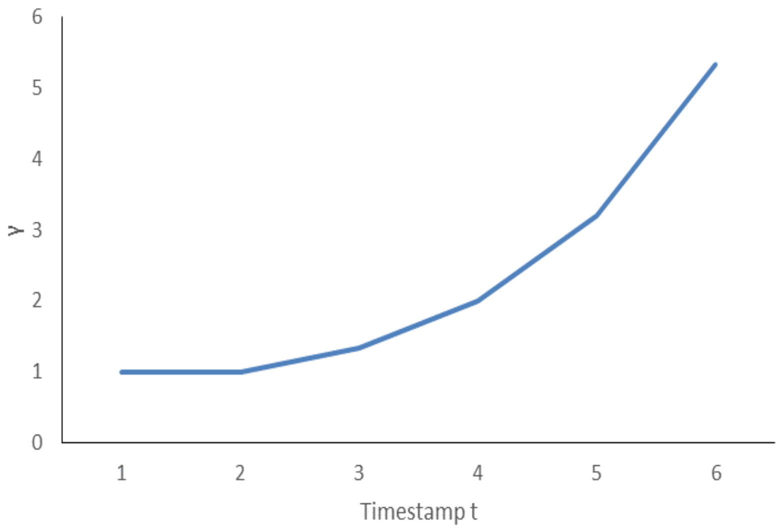


Figure 5.48: γ factor value on each timestamp t.

order to calculate the reward of actions taken in a long-tern data transmission, (5.26) should be a linear function.

Regarding the policy the system follows, it consists in choosing the action will offer the maximum reward. The AI module manages a two-dimensional array like the one shown in Fig. 5.47. When an action has to be taken, the algorithm checks which action, in the current state, has the maximum value, i.e., the reward with the highest value in the row of the current state. This will be the action to carry out. When the reward of this action is obtained, its value is replaced in the array. Thereby, the algorithm is always observing the effects of the actions and learning the best solution to the given problem. The policy function can be described by using (5.27).

$$\pi(s) = a_i | H(s, a_i) \geq H(s, a_j) \quad \forall j \in [1...M] \tag{5.27}$$

, where M is the number of actions defined in the system. Finally, to understand the goal of the system, we should know the two possible points of view: (1) From the point of view of the states, the goal is to remain the maximum time in a non-transmission-problem state. (2) From a reward-based point of view, the main goal of the system is to obtain the maximum reward possible in each timestamp. Therefore, the policy of the system is to choose the action with highest expected maximum value. If we define the Q-value function under policy π, we obtain (5.28):

$$Q_{\pi}(s,a) = \sum_{t=0}^{\infty} MaxR(H,s,t) \tag{5.28}$$

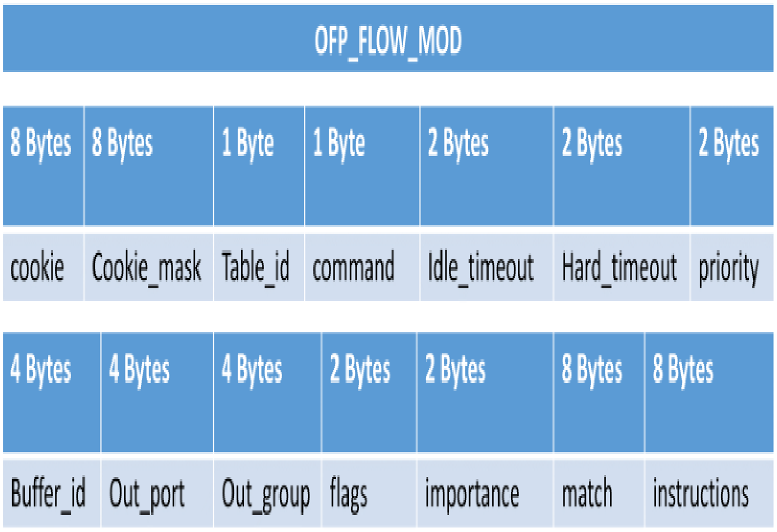
Next subsection defines how the different actions can be performed by using the Openflow protocol.

Actions

This subsection defines the actions described in the Subsection 5.5.3 subsection, and the message exchange using the Openflow protocol. It is assumed that the actions that involve modification in flows will be only performed on those flows belonging to the multimedia transmission.

All the proposed actions can be executed by using the OFPT_FLOW_MOD message. This message allows modifying or deleting the existing flows in a flow table. Moreover, it allows adding new flows. Its structure, explained in chapter 3, is also shown in Fig. 5.49. The more relevant fields for the functioning of our proposal are the command and instructions fields. The command field indicates which action must be done in the table. Adding a new flow, deleting, or modifying an existing one are actions performed by the same OFPT_FLOW_MOD message. The instructions are the actions that must be performed when a flow of packets match with the commands. Those instructions enable us to modify the state of the environment. Depending on the required action, the value of the 'command' field and the instructions vary. For each action defined, the instructions needed and the value of the command field are commented.

The first/second action is to switch to a backup line. The backup line is a dedicated link between nodes that is only used when required. The controller must send an OFPT_FLOW_MOD message to the node that connects with this backup line. In this message the "command" field gets the OFPFC_MODIFY value, and the instruction is OFPAT_OUTPUT. The instruction consists in forwarding packet through the port of the backup line. Therefore, the multimedia transmission will be sent through this backup line without the interfering of other traffic. This, combined with an adequate network configuration, will ensure an acceptable bandwidth or delay in that section of the network.



OFP_FLOW_MOD

8 Bytes	8 Bytes	1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes
cookie	Cookie_mask	Table_id	command	Idle_timeout	Hard_timeout	priority

4 Bytes	4 Bytes	4 Bytes	2 Bytes	2 Bytes	8 Bytes	8 Bytes
Buffer_id	Out_port	Out_group	flags	importance	match	instructions

Figure 5.49: OFP_FLOW_MOD message structure.

The next action is to switch to a backup node. This means that there are one or more network nodes that are going to be bypassed. In order to perform this action, the flow entry related to this flow must be modified. The output port will be changed to the corresponding port that connects with the backup node. In order to perform the action, an OFPT_FLOW_MOD message is sent to the node connected to the backup node. This message is similar to the one sent in the last action, but the output port is the corresponding port to get to the backup node. An OFPT_FLOW_MOD with the command set to OFPFC_ADD and the instruction OFPAT_OUTPUT must be sent to the backup node. Therefore, the backup node will be able to forward the packets to the next node. The output port corresponds to the one that links with the next node. The next node is the same that performing the action before.

Load balancing is a typical technique that takes advantage of the network redundancy in order to ensure a good level of quality in the multimedia transmission. The load balancing can be performed in different ways. The flow that consumes more bandwidth could be forwarded through the redundant secondary way alone. Another solution would be separate the flows attending to their transport layer protocol. Moreover, the multimedia transmission could be forwarded to both paths, changing the route periodically. In this proposal, the multimedia flow is forwarded to an alternative/backup path. Consequently, an OFPT_FLOW_MOD with the command set to OFPFC_MODIFY and the instruction to OFPAT_OUTPUT is sent to the last node in the original path. The output port is the one that connects to the first node in the alternative path. In addition, an OFPT_FLOW_MOD with the command set to OFPFC_ADD and the instruction to OFPAT_OUTPUT for each node in the backup path is sent. Thereby, the flow is able to follow the path and get the destination.

Another possible action is to use QoS-Guaranteed Queues. These queues are high-priority queues that the network nodes can use to forward packets minimizing the delay and jitter of the multimedia transmission. The action is performed again with the OFPT_FLOW_MOD message, but, in this case, the instruction needed is OFPAT_SET_QUEUE. This Openflow action provides queue selection. The OFPT_FLOW_MOD message is sent to the required nodes where the multimedia problem is located, and the queue is set to ensure the QoS required.

The QoS can be handled by using the IEEE 802.1p standard managing the VLAN to differentiate different kind of traffics. Therefore, traffic flows can be managing attending to their requirements. In order to perform this task in our system, the OFPT_FLOW_MOD message uses the OFPAT_PUSH_VLAN instruction. This instruction allows the controller to push a new VLAN tag to the flow. In this tag, the PCP field is indicated. Depending on the value of the PCP field, and the VLAN configuration, the flow can be guaranteed to have a minimum level of QoS.

The last action defined in the system is to limit the available bandwidth of a flow. The Openflow 1.5.1 standard defines meters. Per-flow meters enable OpenFlow to implement rate-limiting. It is a simple QoS operation that constrains a set of flows to a chosen bandwidth. In this case, an OFPAT_METER instruction must be set in the OFPT_FLOW_MOD message. The meter can be configured to constraint either the kbps or the packet/s. In terms of messages, this action can be executed in several ways. On the one hand, the flows that do not contain multimedia content can be modified. An OFPT_FLOW_MOD message will be sent with the OFPAT_METER action as an instruction. On the other hand, the OFPT_FLOW_METER instruction can be added as the default instruction. That means that the default flow will contain the meter instruction. The default flow is the one that will match any flow that does not match with the previous entries in the flow table. How this action is implemented is not critical. However, it is important to know the effect of taking this action and it is well-performed.

As regards the suggestion done before (changing video parameters), the implementation is application specific. The multimedia video server has to provide some API so that the SDN controller can change the video source parameters.

Algorithm

Fig. 5.50 shows the operation algorithm of the Action Chooser which can be divided into four parts, although Fig. 5.50 only shows three of them. Firstly, the algorithm starts initializing the connection with the other modules (Routing, Communication and the Statistic Analyzer) and the data structures (H and M). Then, it listens to changes from both, the Routing Module and the Statistic Analyzer. The secondly, the Action chooser should manage the multimedia flow. To do that, the routing module can notify that a new multimedia flow, called m, has been created. Then, the row for that flow m in the M array is initialized. Attending to the data the routing module has provided, the flags are set. The state and the timestamp are set to zero. If the routing module notifies the end of the multimedia transmission, called m, the data from the M array for that flow is deleted. This is done to maintain the memory efficiency. The third part is the action execution. When the Statistic Analyzer notifies that a multimedia flow m presents a problem in the states, the action should be executed. The state of that flow is updated in the M array and the timestamp and the reward are initialized. The timestamp is set to T, being T the number of iterations for calculating the reward. Then, an action is executed following the policy described in the previous subsection.

The fourth task of the algorithm is the reward calculation in each timestamp. This is performed in an independent thread. Fig. 5.51 shows the operation algorithm of the fourth task. This process is performed for each multimedia flow m with timestamp different to zero. For each timestamp the reward is calculated using the measurements obtained from the Statistic Analyzer. The reward is added to the total reward and the timestamp of the flow is discounted by one. When the timestamp reaches zero, the reward in the H array is updated and the reward is set again to zero. Both arrays are critical sections in the code and the concurrency has to be properly managed in order to ensure the system reliability.

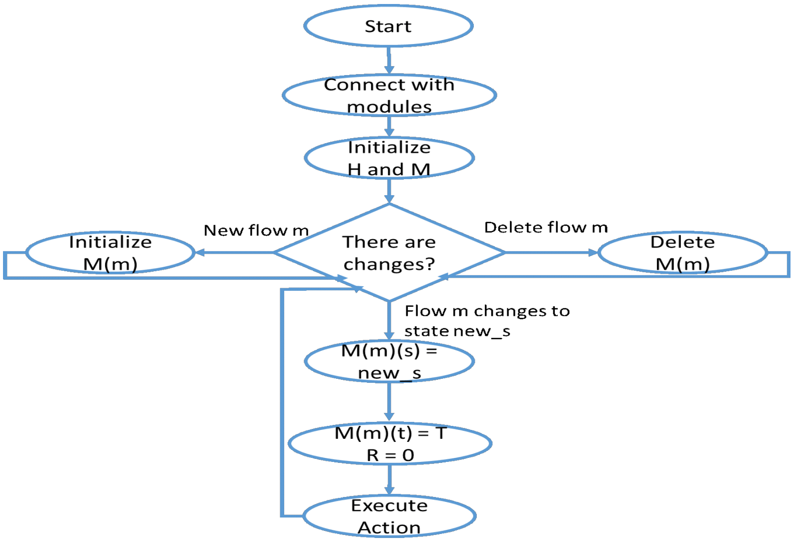


Figure 5.50: General AI algorithm.

Comparison with other proposals

To sum up, Table. 5.2 shows a comparison of our proposal with some of the proposals discussed in chapter 2. Every proposal uses the statistics gathered from the links to choose the best action or route. Although our proposal does not use the characteristics of the multimedia traffic to perform the actions, the routing module could use them to choose the initial path. Only Kassler [96] uses several characteristics. Awobuluyi [92] uses codec layers and Nam [100] the resolution of the video. The actions their proposals perform are oriented to load balancing, changing the route dynamically. However, our proposal defines a set of actions and it decides, based on previous rewards, which action perform. These actions are not restricted except for the limitations of the southbound interface.

Regarding the scenarios on each proposal can be used, our proposal is the only one that fits any scenario. Furthermore, it is the only one that uses AI techniques to learn. This means that it is the most adaptable proposal, and it can be used on every kind of network and conditions. The AI algorithm allows the system to learn the best actions to perform and they may vary according to the scenario. Moreover, the reinforcement learning avoids any supervised learning to be performed.

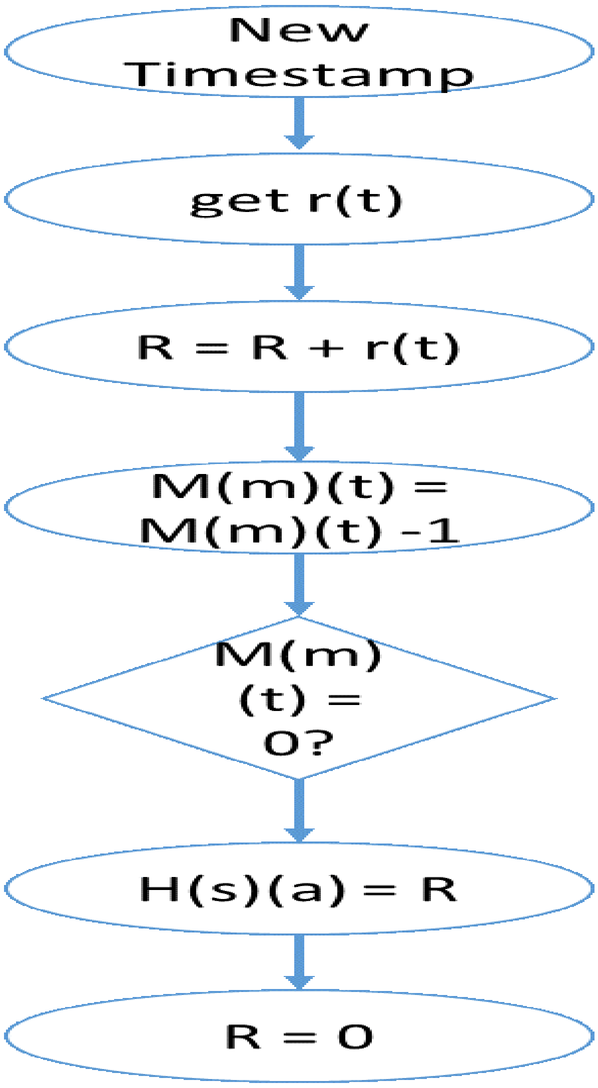


Figure 5.51: Reward calculation algorithm.

Table 5.2: Comparison of the proposals.

	Awobuluyi [92]	**Nam [100]**	**Kassler [96]**	**Our Proposal**
Link stats used	Yes	Yes	Yes	Yes
Video stats used	Video Layers	Resolution	Yes	No
Actions taken	Route changes	Route changes	Route changes	Any Openflow action
It can be used on...	Networks with multiple paths	Networks with video servers	Session oriented traffic	Any Network
Applied to...	Multimedia traffic	Video streaming	Any kind of traffic	Any kind of traffic
AI?	No	No	No	Yes

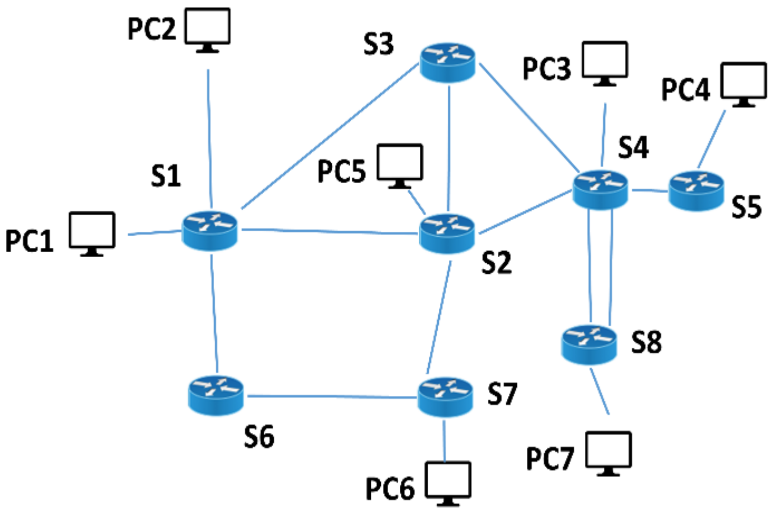


Figure 5.52: Network tested.

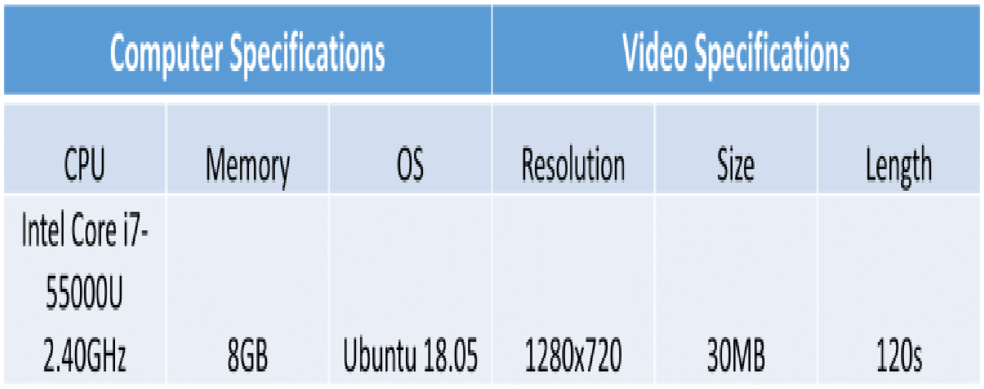
5.5.4 Results

This section analyzes the model proposed. To do it, several scenarios are proposed, and the evolution of the measurements done by the AI module is studied. Moreover, the QoS parameters of the multimedia transmissions are also measured.

Scenarios and Equipment

In this subsection, the scenarios and the network tested are presented. Furthermore, the equipment used to perform the tests is detailed.

The network tested is the one presented in Fig. 5.52. The hosts are labeled as PC1 to PC7 and the network nodes are tagged as S1 to S8. This network presents some characteristics that can be used to test the proposal. First, there are multiple paths to reach the destination. For example, a packet sent from PC2 to PC6 can go through S3 or S6. Moreover, there is a backup line between S4 and S8. All the links are set to 100 Mbps. However, the route chosen by the routing protocol to reach PC7 from S1 is through S3. This is because the link between S6 and S7 presents a higher jitter. All the hosts can work as a multimedia client or as a server.



Computer Specifications			Video Specifications		
CPU	Memory	OS	Resolution	Size	Length
Intel Core i7-55000U 2.40GHz	8GB	Ubuntu 18.05	1280x720	30MB	120s

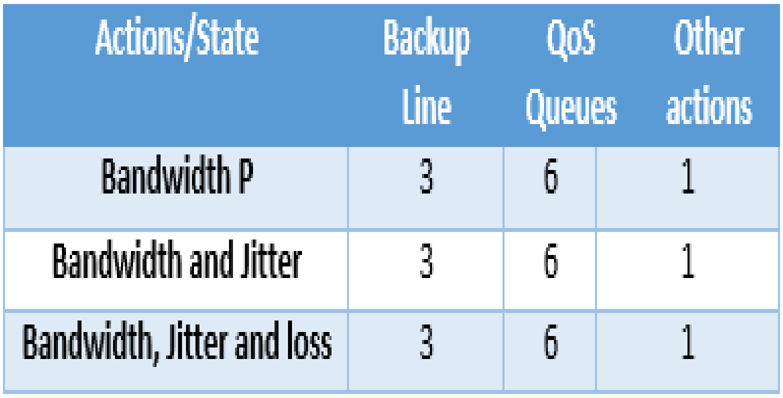
Figure 5.53: Equipment characteristics.

This network is emulated on Mininet SDN emulator. The characteristics of the equipment used to run the emulations are shown in Fig. 5.53.

With this environment, the following tests are performed. First, the reward values are measured depending on the value of τ_{rew} defined in (5.22)(5.22). Then, a comparative of the performance of a multimedia transmission between using the algorithm or not is presented. This comparison is done in two different situations. The multimedia transmission of the tests is a video transmission. The video characteristics are also detailed in Fig. 5.53. For all tests, a QoE evaluation performed by 12 different users is performed. They give a score from 1 to 10 depending on the video quality they perceived to finally calculate the average value. This value is named MOS. The age of the users goes from 8 years old to 86 years old. Moreover, their experience with videos is different from one to each other. Two users have little experience with videos, two users work with video and two are studying university degrees related to new technologies. The rest have experience as video consumers.

Results of the first test

In the first scenario, the video is transmitted from PC1 to PC6. In this case, the node S2 is collapsed and that makes hard to ensure the QoS of the multimedia transmission. In this scenario, the time between iterations and the number of iterations to calculate the reward are changed to evaluate its impact. Seven different transmissions are evaluated. The value of τ_{rew} is changed from 10s to 80s. A value of 70s for of τ_{rew} is not evaluated because the maximum



Actions/State	Backup Line	QoS Queues	Other actions
Bandwidth P	3	6	1
Bandwidth and Jitter	3	6	1
Bandwidth, Jitter and loss	3	6	1

Figure 5.54: H array used in the test.

number of iterations is set to 6. The array H is initialized with the values shown in Fig. 5.54 so that the algorithm has to recalculate the rewards in several iterations. Regarding presentation, not all states and actions have been included in the figure, only the relevant ones in this test.

The action that could solve the transmission problem is to change to an alternative path because S2 is collapsed, regardless the higher jitter presented in the alternative route. However, the initial values will cause the system to choose a useless action in this case. Thereby, the effect that the timestamp parameters have in the QoS during the learning is studied in the first scenario. The video is transmitted for 120s. Fig. 5.55 and Fig. 5.56 show the bandwidth evolution obtained in each transmission. The transmissions have been spitted into two figures attending to the value of τ_{rew}. The transmissions with a value of τ_{rew} from 10 seconds to 40 are presented in Fig. 5.55. Fig. 5.56, the transmissions with a τ_{rew} of 50, 60 and 80 seconds are depicted. The problems in the node limit the bandwidth of the transmission. When the algorithm recalculates the reward and chooses the correct action, the bandwidth consumed by the video transmission increases. The results are summarized in Table. 5.3. The minimum bandwidth is 0.97 kbps in all the transmissions. The maximum bandwidth is quite similar in the first five transmissions, over the 22Mbps. However, the last two transmissions have a maximum bandwidth of 11.95 Mbps for 60s and only 5.70 Mbps for the 80 s of τ_{rew}. The average bandwidth in the first transmissions is also greater than in the last ones. For the 10s and the 20s, an average bandwidth of 9.60 Mbps and 13.30 Mbps have

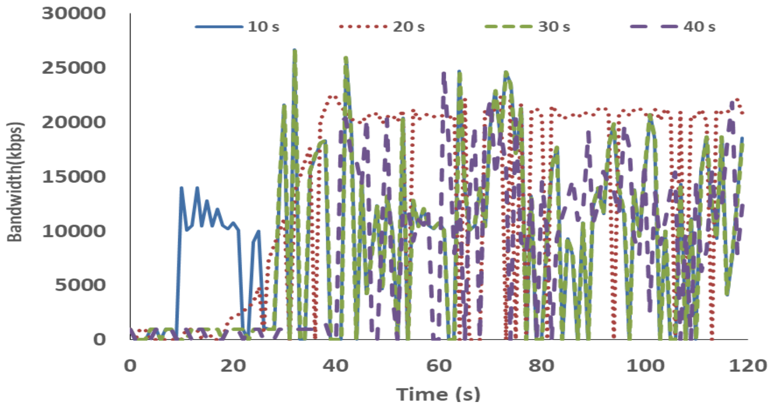


Figure 5.55: Bandwidth evolution in the first test (τ_{rew} from 10 to 40).

Table 5.3: Bandwidth statistics for each scenario.

τ_{rew}	**Min**	**Max**	**Average**
10s	0.97	26686.08	9601.61
20s	0.97	22531.30	13203.28
30s	0.97	26686.08	8399.98
40s	0.97	24733.44	7835.62
50s	0.97	25926.72	7477.58
60s	0.97	11954.50	4873.49
80s	0.97	5706.04	1614.07

been obtained respectively. Nonetheless, for 60s and 80s, an average of 4.87 Mbps and 1.61 Mbps have been gathered.

Fig. 5.57 and Fig. 5.58 show the evolution of the delay obtained in each transmission. Fig. 5.57 shows the transmissions for a value of τ_{rew} from 10 to 40 seconds. On the other hand, Fig. 5.58 shows the same parameter for the transmissions with a τ_{rew} value from 50 to 80 seconds. The minimum delay obtained in all the cases is almost 0. The maximum delay for the last transmission is 20.03ms. For the 30s transmission, it is 30.08ms and for the rest, it slightly varies from 22 to 25ms. The results of delay are summarized in Table. 5.4. Regarding the average delay, it gets higher when τ_{rew} gets higher. The average delay for the first transmissions is 2.71ms and 3.09ms for 10s and 20s respectively. However, with 80s, it is increased to 6.96ms.

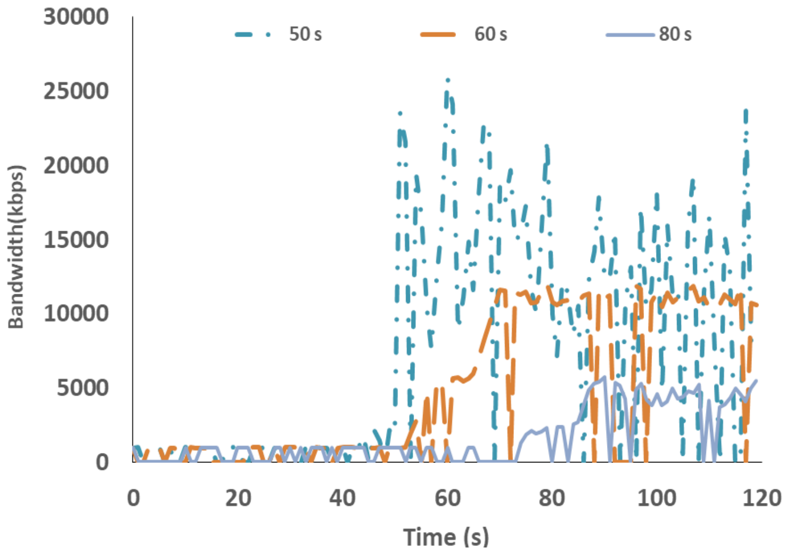


Figure 5.56: Bandwidth evolution in the first test (τ_{rew} from 50 to 80).

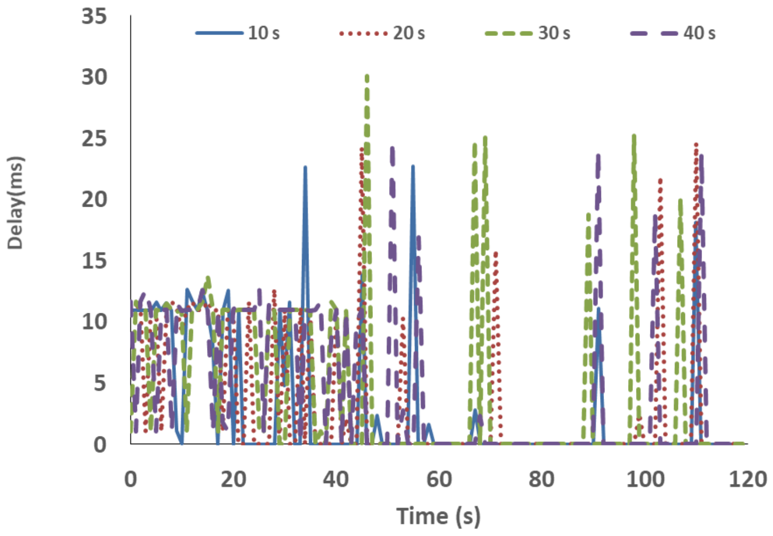


Figure 5.57: Delay evolution in the first test (τ_{rew} from 10 to 40).

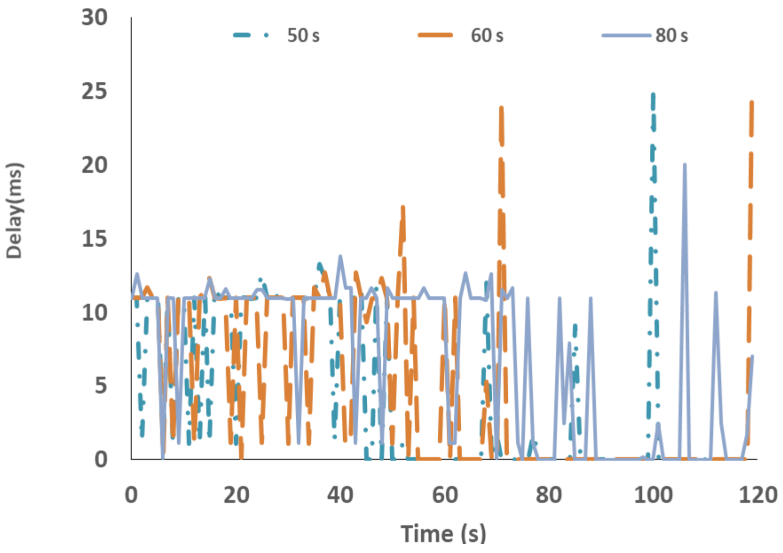


Figure 5.58: Delay evolution in the first test (τ_{rew} from 50 to 80).

Table 5.4: Delay statistics for each scenario.

τ_{rew}	Min	Max	Average
10s	$3.33 * 10^{-17}$	22.71	2.71
20s	$3.78 * 10^{-17}$	24.56	3.09
30s	$3.35 * 10^{-17}$	30.08	4.14
40s	$3.03 * 10^{-17}$	24.81	4.26
50s	$9.69 * 10^{-19}$	25.31	4.13
60s	$2.79 * 10^{-17}$	24.86	4.75
80s	$3.63 * 10^{-17}$	20.03	6.96

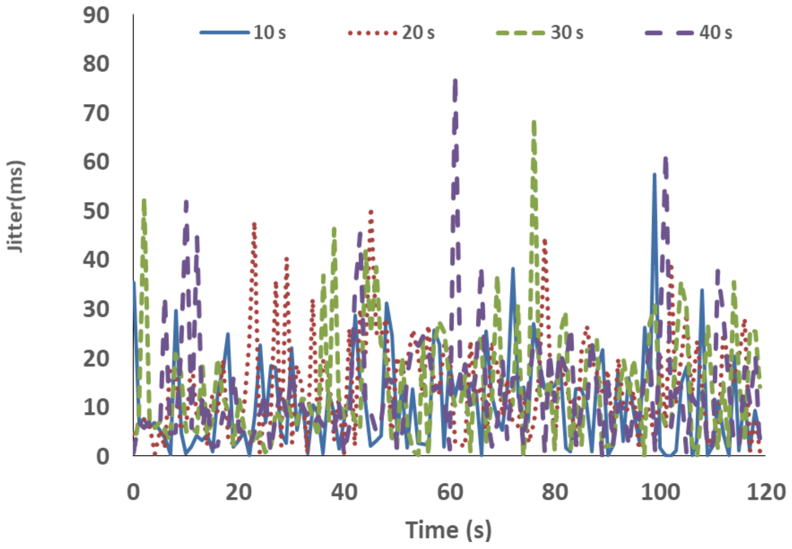


Figure 5.59: Jitter evolution in the first test (τ_{rew} from 10 to 40).

Fig. 5.59 and Fig. 5.60 show the values of the jitter obtained during the transmission. Like Fig. 5.57, Fig. 5.59 shows the transmissions for a value of τ_{rew} from 10 to 40 seconds, while Fig. 5.60 shows the same parameter for the transmissions with a τ_{rew} value from 50 to 80 seconds. The minimum jitter obtained barely varies from the different transmissions. Except from the 20s and 40s transmissions, it stays in values close to 0ms. For these transmissions, the minimum jitter values are 0.55 and 0.14 respectively. The 60s and 80s transmission have the lowest minimum jitter values. However, regarding maximum jitter they have the highest values after the 40s transmission. It has a maximum jitter of 77.89ms, while 60s and 80s transmissions have 69.97ms and 72.84ms respectively. The average jitter varies from 10.73 for the 10s transmission to 14.08 for the 20s transmission. The transmissions with 60s and 80s of τ_{rew} have an average jitter of 11.43 and 12.94ms respectively. The results of jitter are summarized in Table. 5.5.

Regarding the loss rate, the results for each transmission are shown in Fig. 5.61. With 10 seconds, the loss rate is 0.3% and with 20 seconds 0.6%. From 30 seconds to 60 seconds, the loss rate increases from 7.1% to 8.1% respectively. The loss rate of the 40 seconds scenario is 7.9% and 5% from the 50 seconds scenario. Finally, the highest loss rate is obtained when the recalculation takes 80 seconds to be done. This loss rate is 14%. This is caused by the collapse

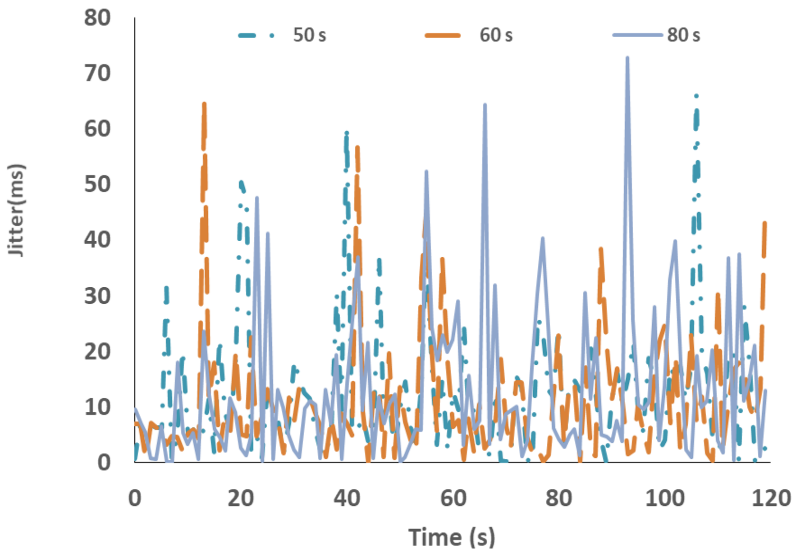


Figure 5.60: Jitter evolution in the first test (τ_{rew} from 50 to 80).

Table 5.5: Jitter statistics for each scenario.

τ_{rew}	Min	Max	Average
10s	0.01	57.43	10.73
20s	0.55	50.69	14.08
30s	0.01	68.73	13.75
40s	0.14	77.98	13.15
50s	0.02	66.20	11.96
60s	$7.63 * 10^{-16}$	69.97	11.435
80s	$1.00 * 10^{-15}$	72.84	12.94

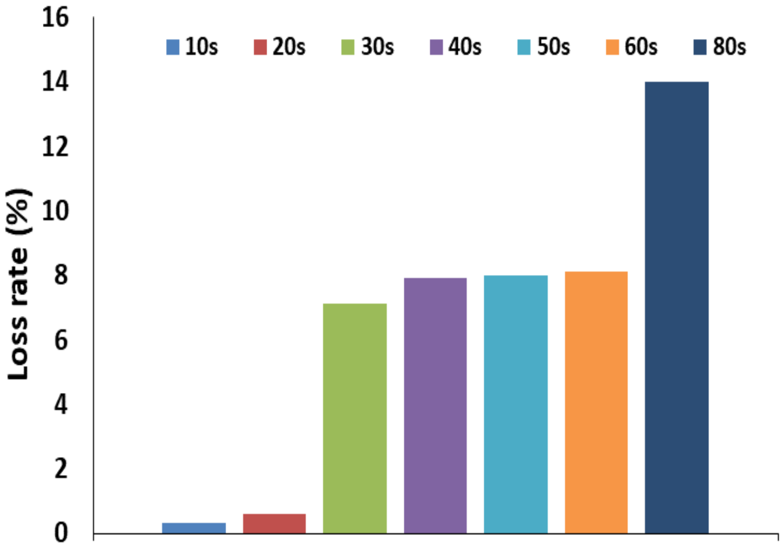


Figure 5.61: Loss rate for each scenario.

in the node. The node S2 is not able to forward all the data packets and drop many of them. Changing to the backup line earlier reduces the packets drop by S2.

Finally, the QoE is measured by gathering the MOS of the users. Fig. 5.62 shows the value of MOS obtained from the opinion of the users. For the transmission with 10 s of τ_{rew}, the score obtained is 7.66 of 10. The 20s scenario has received a 6.85 and the 30s a 6. A score of 5.12 and 4.89 has been given to the scenarios with 40 and 50 seconds of τ_{rew}. With 60 seconds of τ_{rew}, a MOS of 4 has been obtained. Finally, the last scenario has received a 2.75 score. The QoE experienced by the users is lower when τ_{rew} gets higher. This is related to the high loss rate, which causes problems in the transmission such as frozen frames, tiling, noise, ghosting, soft focus, and flickering.

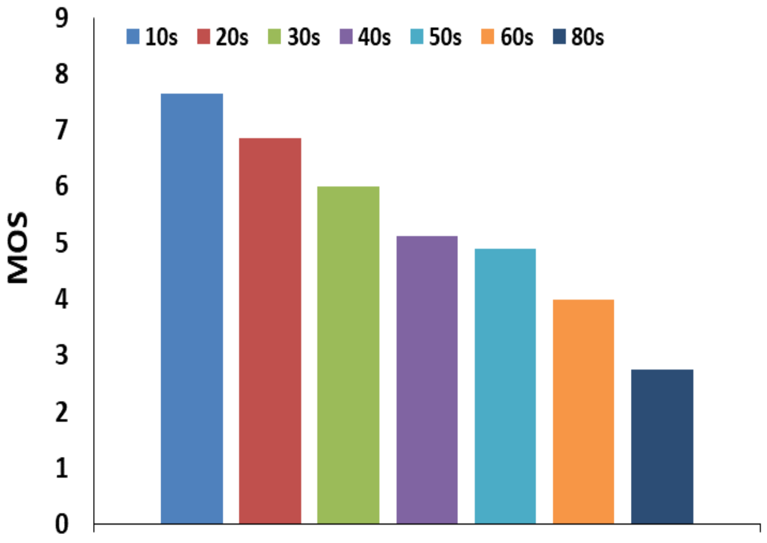


Figure 5.62: MOS obtained in each scenario.

Results of the second test

In the second test, the video is transmitted again from PC1 to PC6 in similar conditions to the previous scenario. However, in this test, the difference in terms of QoS and QoE between using the proposal and not taking any action is measured. The transmission with the algorithm has a τ_{rew} of 40s. This value has been chosen to be neither too low nor high. Therefore, it is not an optical parameter value, but it provides some improvement to the transmission.

Fig. 5.63 shows the consumed bandwidth during the video transmission. As we can see, when the algorithm is not used, the minimum bandwidth registered is 0.97 kbps while the maximum value is 987.16 kbps. The average bandwidth during this transmission is 412.42 kbps. On the other hand, when the proposed algorithm is used, the maximum value of bandwidth is 24.73 Mbps while the average bandwidth is around 7.83 Mbps.

The delay registered during the video transmission is shown in Fig. 5.64. The transmission without the algorithm presents a maximum delay of 13.32ms and an average of 8.94ms. However, using the algorithm, the maximum is 24.81ms and the average is 4.26ms. Without the algorithm, the delay relays more stable, although higher. Nevertheless, using the algorithm, the transmission suffers

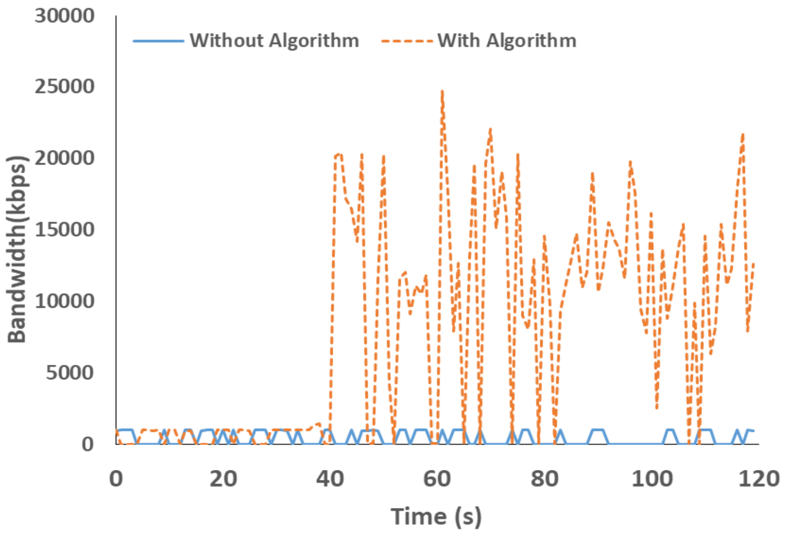


Figure 5.63: Bandwidth evolution in the second test.

some delay peaks. This is translated into some little video errors like ghosting and black pixels during the transmission.

Regarding the jitter, Fig. 5.65 shows the difference between the two transmissions for both cases, i.e., when the algorithm is used and when it is not used. Without algorithm, the maximum jitter is 78.76ms and the average is 12.34ms. With the algorithm, the maximum jitter is 77.98ms and the average 13.15ms. Both transmissions have a stable jitter, with some peaks Therefore, both jitter graphs are quite similar, with only an increment of 6% of the average jitter. Furthermore, the jitter peaks produced when the algorithm is used are due to the lower performance of the alternative route.

The loss rate is also compared for both cases. Fig. 5.66 shows this comparison. In blue, the transmission when the algorithm is not used presents a 20.5% of loss rate. Nevertheless, when using the algorithm, the loss rate is reduced up to 7.9%.

Regarding to the QoE results (see Fig. 5.67), the average value of all the 12 requested users shows that when the algorithm is not used, the MOS has a value of 1.66 points over 10 while using the algorithm improves in great measurement the results with a value of 5.12 over 10 points. This shows the increment of performance that the action taken by the algorithm provides. Despite the

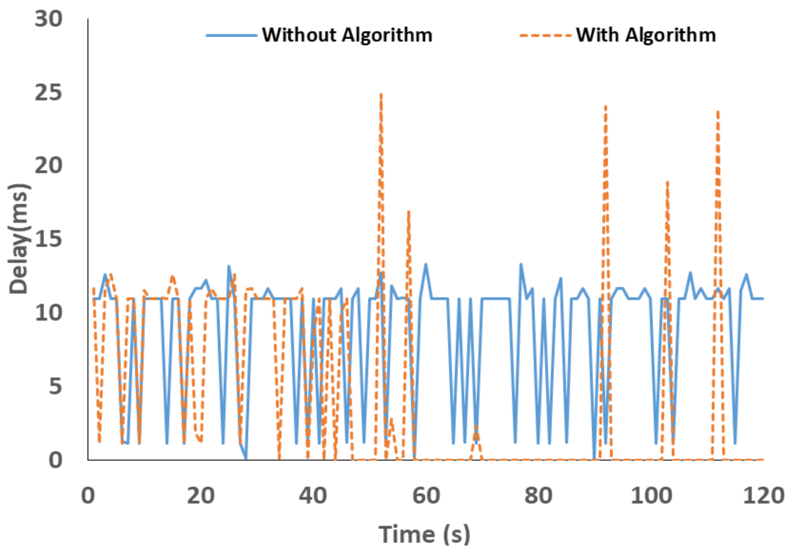


Figure 5.64: Delay evolution in the second test.

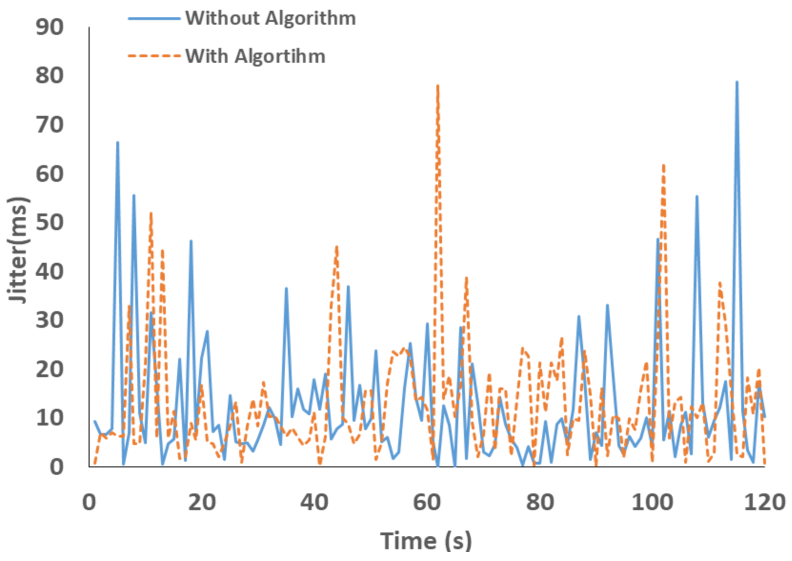


Figure 5.65: Jitter evolution in the second test.

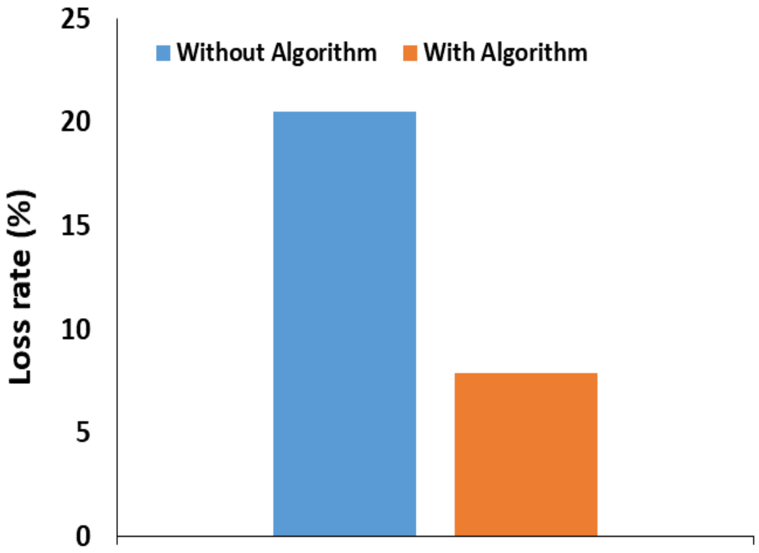


Figure 5.66: Loss rate obtained in the second test.

lower performance of the alternative route, the quality is increased when the problem is handled by the algorithm.

5.5.5 Conclusion and Future Work

Multimedia transmissions require an important availability of resources to ensure an acceptable quality. So, the way the networks manage their resources have a critical impact on this quality. SDN allows managing the resources in a more efficient way. Using AI, networks can be aware of the problems and learn how to solve them in order to provide the best QoS and QoE in multimedia transmissions. Machine learning is a technique that fits the network resource management problem. So, taking into account the aforementioned issues, in this subsection, we have presented an adaptation of this technique to SDN. The results obtained show that the introduction of the proposed algorithm improves the quality of the multimedia transmission. In terms of QoE, users perceive an increase in the image quality three times better, while the loss rate is reduced more than half the value of losses recorded when the algorithm is not applied. Regarding bandwidth, the maximum throughput increases from 987.16 kbps to 24.73 Mbps while the average bandwidth improves from 412.42 kbps to 7.83 Mbps.

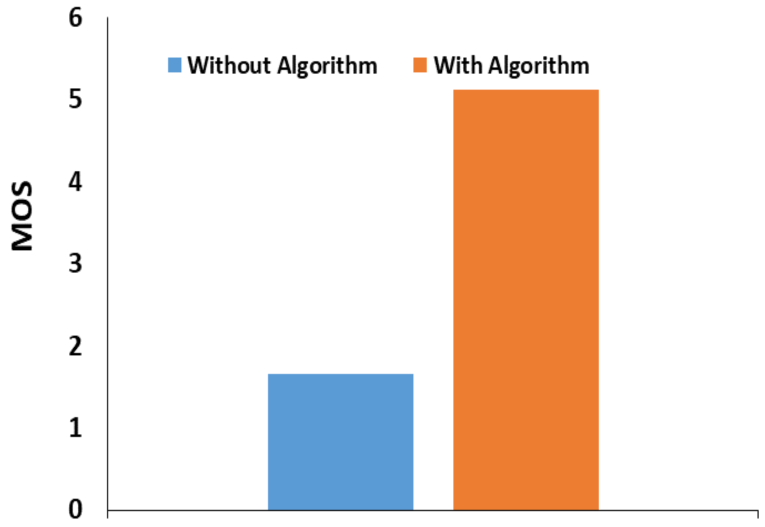


Figure 5.67: MOS gathered for the second test.

Although these results show interesting improvements in multimedia transmission, some considerations should be taken into account. Firstly, the proposed system is a preliminary solution to these problems. The parameters defined are used in the system, but their values may change to adapt the system to the problem. Depending on the problem or the evolution of the Openflow protocol, the states and actions the system manages may vary. Furthermore, the parameters that control the total amount of time needed to recalculate the rewards can have different values. This affects the quality of the multimedia transmission as the results of the first scenario show. The MOS of the transmission with quicker reactions to the problem are higher, i.e., from 7.66 with the quickest reaction to 2.75 with the slowest one. Furthermore, the average bandwidth of the transmission is high. With 10 or 20 seconds of τ_{rew}, the average bandwidth consumed is between 9 and 13 Mbps. However, increasing τ_{rew}, the system reduces the bandwidth up to 1.6 Mbps. Loss rate also increases from 0.3% and 0.6% to 14%.

5.5.6 Reward Update System

In this subsection, the problem of evaluating the impact of the action chosen by the system will be briefly addressed. As it was described in chapter 4, the routing algorithm enters into an evaluation state when an action is performed. This state is kept during a period of time, delta. Once this time expires, the system must compare the current situation of the network with the one before performing the action.

The comparison can be performed differently, depending on the method use in the AI module. Since the solution proposed in this work is a reinforcement learning method, the evaluation of the output updates the reward in the algorithm. Hence, the evaluation must be accurate. The goal of this evaluation is to know if the increase or decrease of the QoS of the multimedia transmissions are a consequence of the action taken.

This work proposes a static method of validation. Table. 5.6 shows a representation of the data structure used in this method. Depending on the problem the action had to solve, the evaluated metrics will vary. The data structured is built according to the work done by our research group, since it has a vast trajectory in multimedia and networking. Laura G. et al. summarize in [186] the dependencies between multimedia transmission problems and metrics. Miran Taha et al. found a relation between network parameters and metrics problems in video transmission [187]. Along with [188], these works were taken into account to build the static data structure that specifies which metrics should be taken into account to determine if the action performed by the routing module is not the right one. In that case, the reward of that action will be penalized. Otherwise, if the problem has been solved, the reward is increased. Consequently, Table. 5.6 shows which metrics should be taken into account based on the actions performed in the network.

In order to increase the accuracy of the reward update system, the data structure could be create not based on previous investigations, but on a heuristic method, assigning a weight to each metric in each situation. However, this is out the scope of this dissertation.

Table 5.6: Relevant metrics for action evaluation.

	Delay	**Jitter**	**Throughput**	**Packet Losses**
Change to a backup switch	Yes	No	No	No
Route change	Yes	Yes	Yes	Yes
Load balancing	No	Yes	Yes	Yes
Queuing/VLAN management	Yes	No	No	No
Link aggregation (backup line)	No	Yes	Yes	Yes
Limit flow throughput	No	Yes	Yes	Yes

5.6 Routing Alternative Evaluation

In this section, the performance results of the routing proposal are evaluated. Therefore, a comparison between the traditional OSPF routing algorithm, the modification for SDN presented in chapter 4 and the protocol with the reinforcement learning algorithm is performed in this section. First, the network and the equipment used are described. Then, the different scenarios tested and the characteristics of the multimedia traffic sent are detailed. Finally, the results are discussed.

The network topology used is shown in Fig. 5.68. In this network, the core is built over SDN switches that form a ring with a sort of shortcut through the middle of the network. This switch in the middle acts as a backup switch. Each of the switches is connected to a heterogeneous network, where different kinds of clients may be found. All the networks are /24-mask-sized. These networks, depicted in Fig. 5.69, send different kind of traffic through the SDN core network. Consequently, the SDN controller manages the traffic, always prioritizing the multimedia traffic when facing transmission problems. The networks that interconnect the switches, needed for OSPF-based solutions, are shown in Table. 5.7. In Fig. 5.68, when a double link interconnects two switches, it means that there is a backup line that can be used if needed.

In Table. 5.8, the equipment used for the multimedia server and clients, the SDN controller and the SDN switches is specified. That is the hardware used for the test performed in this section. The test is divided into two different

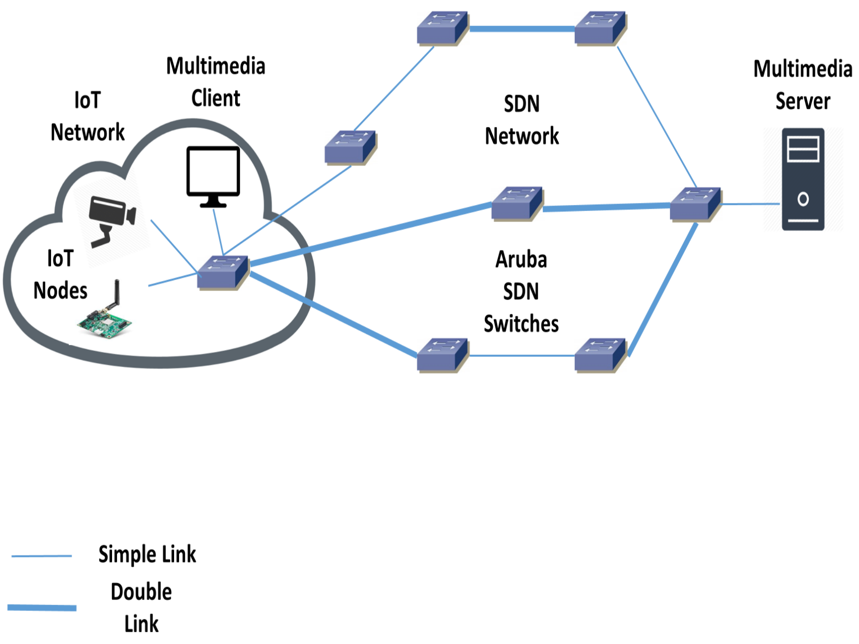


Figure 5.68: Proposed topology.

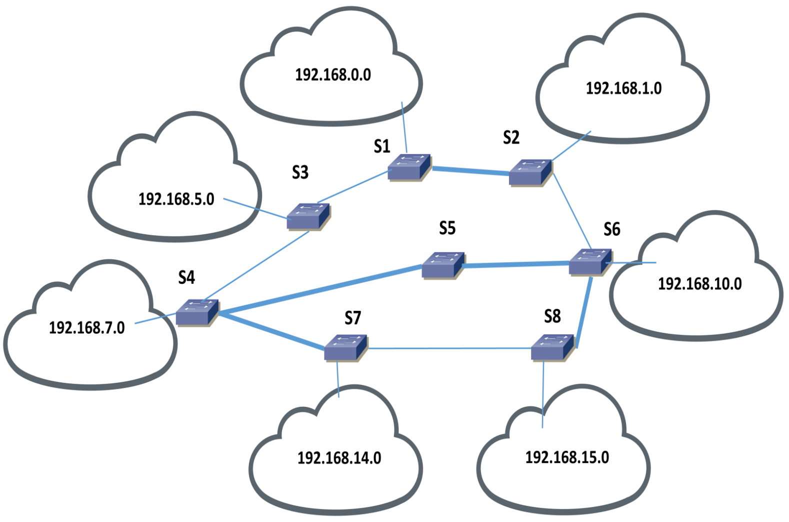


Figure 5.69: Loss rate obtained in the second test.

Table 5.7: Networks between switches.

Network	Source	Destination	Double Link
192.168.2.0	S1	S2	Yes
192.168.3.0	S1	S3	No
192.168.4.0	S2	S6	No
192.168.6.0	S3	S4	No
192.168.8.0	S4	S5	Yes
192.168.9.0	S5	S6	Yes
192.168.11.0	S4	S7	Yes
192.168.12.0	S6	S8	Yes
192.168.13.0	S7	S8	No

Table 5.8: Equipment specifications.

	CPU	RAM	OS
Network Hosts	Intel Core 2 Quad CPU Q800 @ 2.33 GHz	4 GB	Windows 10 Education
SDN Controller	Intel Core i7-55000U @ 2.40GHz	8 GB	Ubuntu 16.04
SDN Switches	Aruba 2930F 24GB 4SFP [189]		

scenarios. Each scenario shows a different multimedia transmission as a focus of the test. For every multimedia transmission, some problems appeared during the streaming. For both scenarios, the video transmitted and the software to transmit and to measure the streaming is the shown in Table. 5.9.

The two different scenarios are discussed in the following subsections, where both the description of the scenario and the results obtained are depicted. In both scenarios, the traditional OSPF routing protocol is compared to the OSPF adaptation to SDN (OSPFMod) and to the reinforcement learning AI method in the routing protocol proposed in this dissertation (RL).

Table 5.9: Software and video used.

Video	Big Buck Bunny, Sunflower MP4 version. 1920x1080, 750MB.[190]
Sreaming Software	VLC media player v. 3.0.11. [191]
Capture Software	Wireshark 3.2.7 [192]

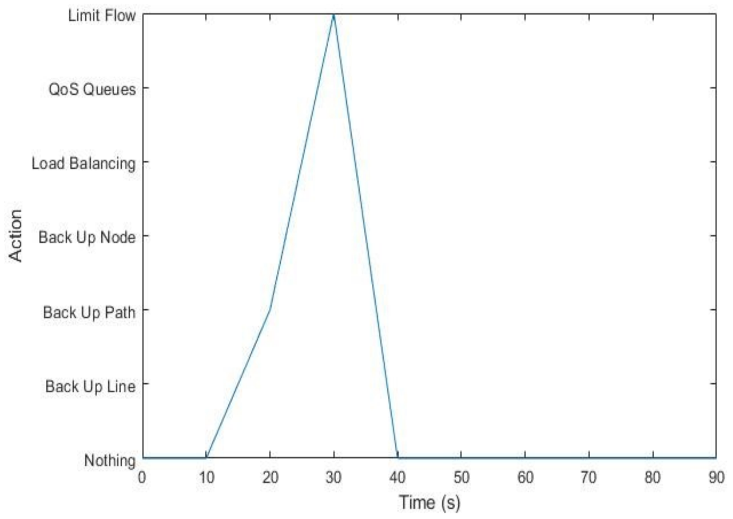


Figure 5.70: Actions selected by the RL algorithm in scenario 1.

5.6.1 Scenario 1

The first scenario evaluates a multimedia transmission from the end network directly connected to S3 (192.168.5.0) to a multimedia host in the network connected to S2 (192.168.1.0). In this scenario, all the protocols initially decide to send the streaming through S1. In this scenario, the transmission has a duration of 90 seconds. However, approximately after 15 seconds, the link between S3 and S1 stop working. Since the route through S5 is a backup switch, all the protocols decide to send the data through S4 and S7. In that route, there is a TCP connection sending huge files and consuming bandwidth. Fig. 5.70 depicts the actions selected by the RL algorithm to be performed in the network. From second 10, the protocol selects an alternative route. In second 30, the RL algorithm decided to limit the TCP flow.

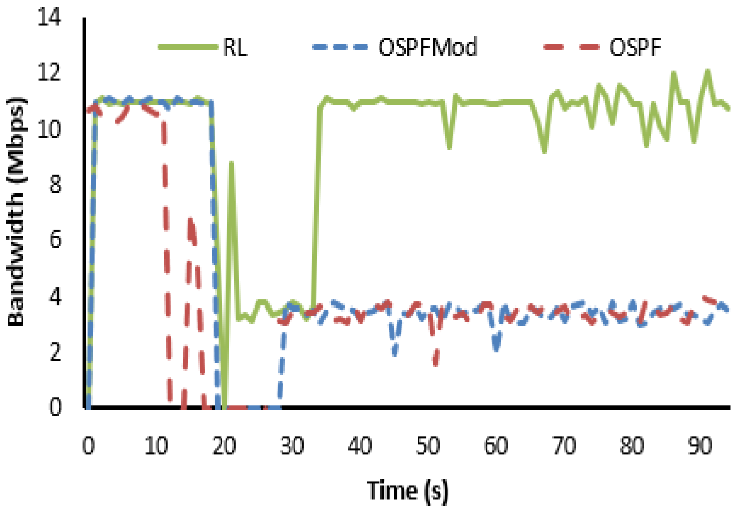


Figure 5.71: Bandwidth consumed in scenario 1.

Fig. 5.71 shows the bandwidth consumed by the multimedia transmission. All the protocols have a minimum bandwidth of 0 Mbps. This fact is due to that, in some point of the communication, all of them lose connectivity. As average bandwidth, OSPF presents 3.859 Mbps. The modification of OSPF has an average of 4.488 Mbps and the proposal 10.783 Mbps. The maximum bandwidth consumed is 10.859 Mbps by OSPF, 11.069 Mbps by OSPFMod and 12.056 Mbps by the proposal.

Fig. 5.72 depicts the delay presented by the multimedia transmission for each protocol. Again, the minimum delay for all protocols is 0ms. However, maximum and average delays vary depending on the protocol. OSPF presents an average delay of 23.32ms and a maximum of 104.21ms. The average delay of OSPFMod is 22.14ms and the maximum 80.48ms. Finally, the proposal has an average delay of 10.10ms and a maximum of 198.89ms.

In Fig. 5.73, the jitter suffered by the multimedia transmission is presented. The minimum values are 0.01 ms for OSPF and OSPFMod and 0.08 ms for the proposal. The average jitter presented by OSPF is 3.22ms and the maximum 23.08ms. On the other hand, the modification of OSPF has an average jitter of 2.79ms and a maximum of 11.13ms. Finally, the average jitter of the proposal is 1.98ms and the maximum is 24.65ms.

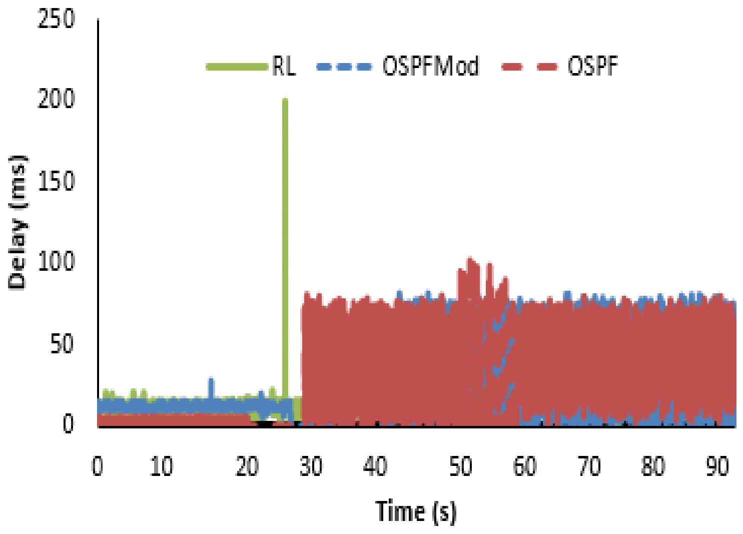


Figure 5.72: Delay presented in scenario 1.

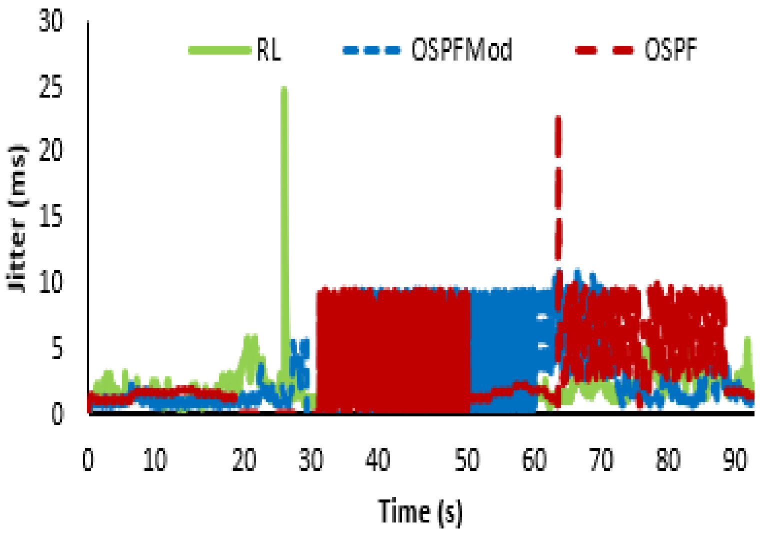


Figure 5.73: Jitter presented in scenario 1.

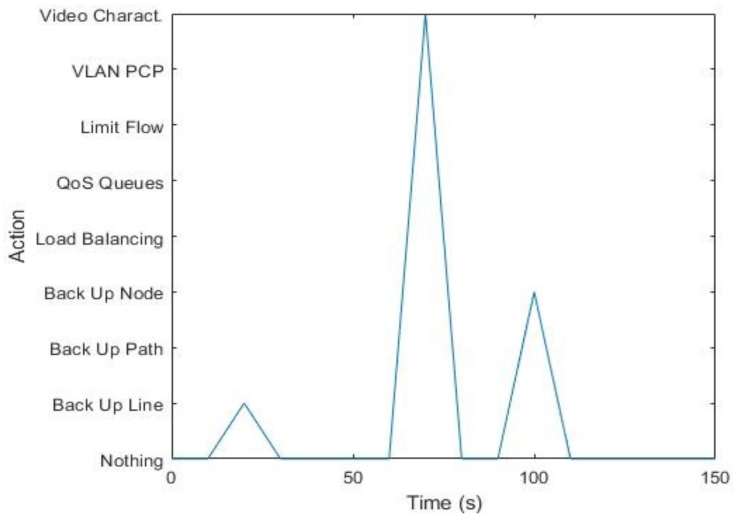


Figure 5.74: Actions selected by the RL algorithm in scenario 2.

5.6.2 Scenario 2

The second scenario evaluates a multimedia transmission from the end network directly connected to S4 (192.168.7.0) to a multimedia host in the network connected to S6 (192.168.10.0). After approximately 15 seconds, the first of the links between S4 and S7 stop working. This produces different behaviors in the protocols. However, once the new path is established, the increment of the traffic in that path makes congestion to happen after some seconds. Then, the different approaches between the protocols play again a role in the performance.

Fig. 5.74 displays a graphical representation of the actions chosen by the RL algorithm. In this scenario, the algorithm decided firstly to change the route. After the congestion, the video characteristics were modified, downgrading the quality of the source to avoid transmission problems. Finally, after the last connectivity problem, the algorithm decided to use the backup switch, S5, to avoid congestion.

Fig. 5.75 shows the bandwidth consumed by the multimedia transmission. Again, due to a lack of connectivity during the streaming, all the protocols have a minimum bandwidth of 0 Mbps. As average bandwidth, OSPF presents 7.497 Mbps. The modification of OSPF has an average of 8.775 Mbps and the

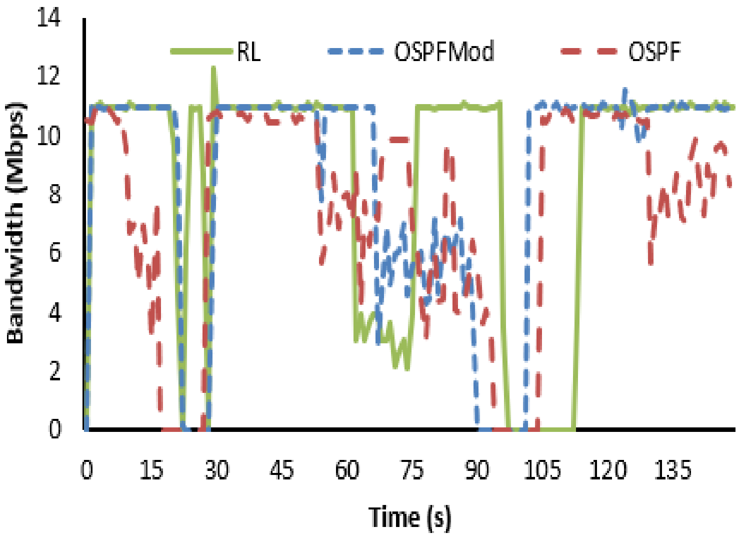


Figure 5.75: Bandwidth consumed in scenario 2.

proposal 8.772 Mbps. The maximum bandwidth consumed is 10.874 Mbps by OSPF, 11.508 Mbps by OSPFMod and 12.275 Mbps by the proposal.

Fig. 5.76 depicts the delay presented by the multimedia transmission for each protocol. Again, the minimum delay is 0ms for all protocols. Nevertheless, maximum and average delays vary depending on the protocol. OSPF presents an average delay of 4.66ms and a maximum of 23.74ms. The average delay of OSPFMod is 4.66ms and the maximum 23.74ms. Finally, the proposal has an average delay of 8.84ms and a maximum of 246.68ms.

In Fig. 5.77, the jitter shows the bandwidth consumed by the multimedia transmission. The minimum values are 0.07ms for OSPF, 0.29ms for OSPFMod and 0.03ms for the proposal. The average jitter presented by OSPF is 3.66ms and the maximum 11.42ms. On the other hand, the modification of OSPF has an average jitter of 0.84ms and a maximum of 23.43ms. Finally, the average jitter of the proposal is 1.80ms and the maximum is 27.55ms.

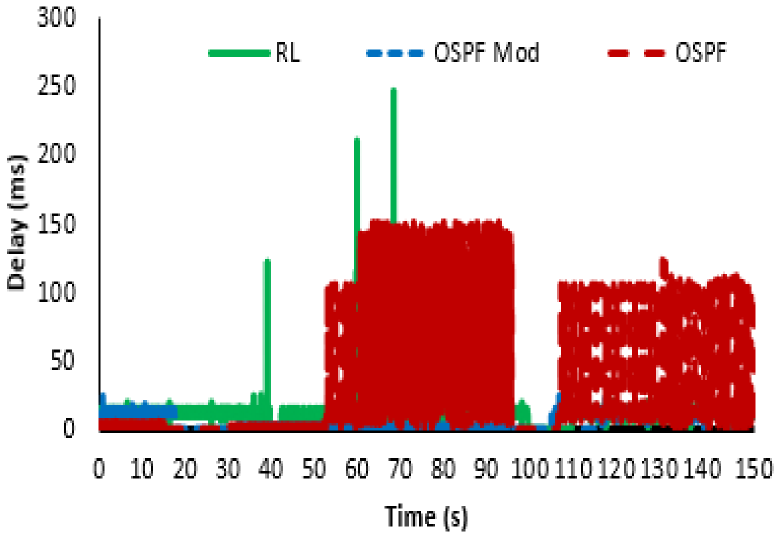


Figure 5.76: Delay presented in scenario 2.

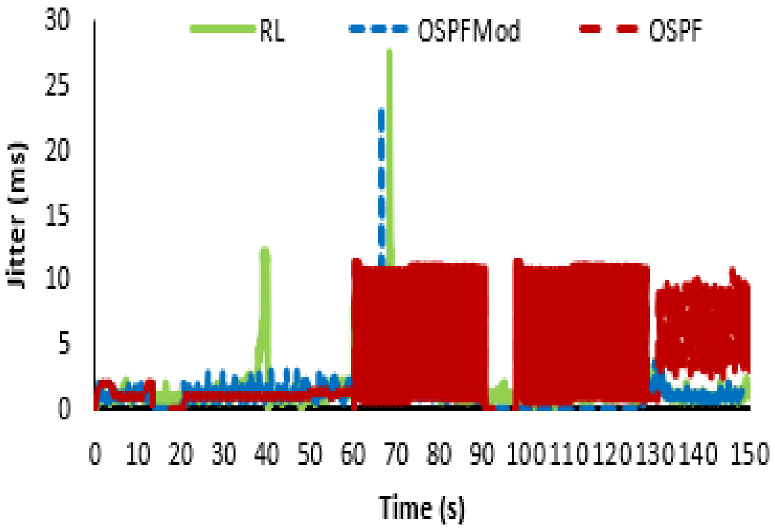


Figure 5.77: Jitter presented in scenario 2.

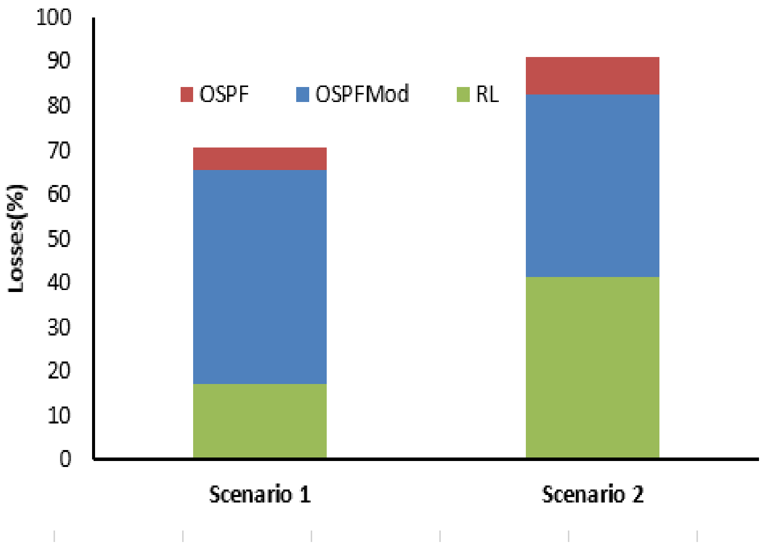


Figure 5.78: Packet losses.

5.6.3 QoE Results

In this subsection, the QoE results are discussed. First, the packet loss of each transmission in each scenario is presented. Then, the QoE measurement is explained and the results are depicted. As regards the packet losses, Fig. 5.78 shows the comparison between the different protocols in both scenarios. In both scenarios, the proposal is the protocol with less packet losses (16.85% and 41.09% respectively). The modification of OSPF is the second protocol with less losses (65.25% and 82.25%). Finally, OSPF presents losses of 70.63% and 90.85%).

The QoE experiments consist in playing the received video of each scenario and protocol to a set of users. Then, the users must score the quality perceived of the video in a scale from 0 to 10. Then, the average is calculated. Therefore, the MOS is obtained. Table 5.10 shows the description of the users that analyzed the video quality.

Fig. Fig. 5.79 shows the results of the QoE measures. For both scenarios, the order of the transmissions depending on the QoE perceived is the same. OSPF has the worst results with, approximately, 6 and 4 respectively. In the second position, we find OSPFMod with 6.25 and 5. The proposal provided better quality with scores of 7 and 6, approximately.

Table 5.10: Users data.

Sample Size	15
$<$ 30 years old	10
$>$ 30 and $<$ 60 years old	4
$>$ 60 years old	1
Not familiar with technology	2
Low familiarity with technology	9
High familiarity with technology	4

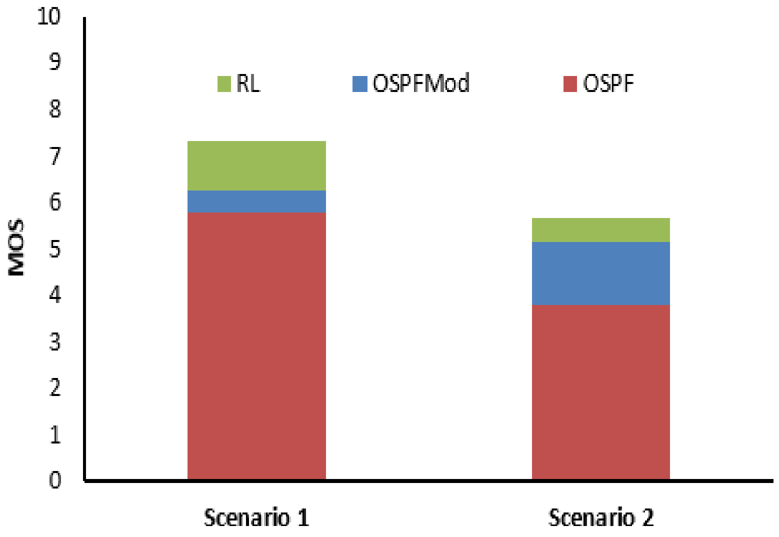


Figure 5.79: QoE results.

5.7 Conclusion

In this chapter, the AI methods used for the proposal have been detailed. All the AI processes included in the AI module located in the architecture has been named and discussed. Finally, a comparison between the routing alternatives has been provided. This comparison shows that, due to the new actions introduced by the proposal, the QoS, in terms of throughput, delay, jitter and packet loses, presents better performance than traditional routing protocols, although they are adapted to SDN. Furthermore, regarding QoE, the results show that the users perceived better quality from the multimedia transmission over the proposal rather than the other alternatives, even although all the transmission suffered problems during the streaming.

Chapter 6

Applications

6.1 Introduction

In the last chapters, the SDN controller, the routing algorithm and the AI module have been discussed. In this chapter, some of the applications proposed that are built from the system, and the architecture previously depicted, are described. These applications have been published in journals, book chapters or conferences (see chapter 7).

6.2 SDN Applied to Smart Irrigation

The proposed system has been designed to enhance the efficiency in the water usage in the smart cities. This efficiency is achieved by two different ways.

The first one, is to adjust the irrigation needs in order to avoid waste of water. It will be done by controlling the exact amount of water need. Commonly, the irrigation process is done according to pre-set values based on the grass type and the season of the year. However, the climatic conditions, the topology, the soil type or the shadow conditions are not considered. Those conditions cause that, in different parts of the same garden, the water demand is different. Thus, the requirements of irrigation change from one part of the garden to another one. For this reason, it is necessary to consider the humidity in the soil before to distribute the water in the gardens. This methodology is widely used for agricultural purposes and it is part of the precision agriculture. Nevertheless, these techniques are not used as often in gardening. We can find some examples in the literature where authors employ sensors to adjust the irrigation in crops [193] or [194].

The second one, is to define a series of rules to ensure that the irrigation is done in the best moment. The most important rule is to water the gardens only during the night periods. This will reduce the fast evaporation of the water, which cause many problems. Moreover, the system considers the water consumption of houses and business in order to decide the best moment to irrigate. The objective is to use water for irrigation purposes in the moment when the houses and business are not using water. Thus, the pressure over the water supplies fonts will be reduced. It is better to have similar water consumptions along the 24 hours of the day, when there are different demand peaks.

For that purpose, our system must be able to read data on the state of the gardens state and on the water consumptions done by houses and business. In addition to read a high amount of information, its integration and pattern identification and prediction is crucial. Therefore, a smart system able to take decisions based on a set of given rules is needed. Moreover, the proposed system needs to ensure the security of the data sent from houses and business and it must be easily scalable due to the continued growth of the cities.

6.2.1 Urban Scenarios

In this section, we are going to identify and describe the urban scenarios where our proposal can be developed. First, the different areas in the cities are defined. Then, the employed sensors in each area are shown.

For our proposal, we are going to differentiate two areas inside the urban environment. Residential areas, where homes and business are located, usually consume high amounts of water during the day and the first hours of night. Water is mainly consumed for domestic purposes, cleaning, cooking and hygiene among others. In small percentage, some houses have their own gardens and use water for irrigation purposes or to fill swimming pools. Water is used in business for almost the same purposes that at home. Depending on the business they can use more or less water and their maximum consumption can occur at different hours.

In green areas, water is mainly consumed for irrigation purposes, but it also can be used in drinking water fountains and decorative fountains. The biggest water consumption occurs during the night when the gardens are irrigated. According to the climatic conditions, grass requirements and soil type, the irrigation needs may change. For irrigation purposes, we can distinguish between walkable green areas and non-walkable green areas. On the one hand, walkable green areas are considered all the green areas where the presence of people is common, like parks, gardens, sport areas or recreational areas. On the other hand, non-walkable green areas are all the green areas where presence of people is not expected like the roundabouts and similar areas. Non-walkable green areas can be watered anytime in the night. In walkable green areas we can distinguish two categories. The first one, areas with opening and closing hours like sport areas can be watered in the night during the closing period. The last one, green areas without opening and closing time, must be watered during the night but ensuring that nobody is there.

As a result, we can have 4 different types of areas in the city, (i) residential areas, (ii) non-walkable green areas, (iii) walkable green areas with no schedule, and (iv) walkable green areas with schedule.

6.2.2 Employed Sensors

In residential areas, sensors to measure the water consumption will be deployed. Those sensors are integrated in the water supply system and they are called smart meters. Smart meters are digital electronic devices that can collect data on supplies use and send it to the utility [195]. The supplies can include water, electricity or gas. In this case the smart meters will monitor the water use.

In green areas, sensors to measure the garden conditions and actuators that activate/stop the irrigation system will be deployed. The sensors that control the garden status include rain (Fig. 6.1), soil moisture (Fig. 6.2), temperature (Fig. 6.3), wind [196] and solar radiation (Fig. 6.4) sensors. Those sensors are located in atmospheric stations placed in the gardens, water flow sensors in the water pipes, and presence sensors (Fig. 6.5). Soil moisture sensors define the amount of water needed by a portion of garden. Temperature, rain, wind and solar radiation sensors are used to define the best moment to irrigate. The current sensors in pipes are employed to ensure that there is no pipe broken. Presence sensors, which are placed along the gardens without special schedule, define, in conjunction with climate sensors, the best moment to irrigate.

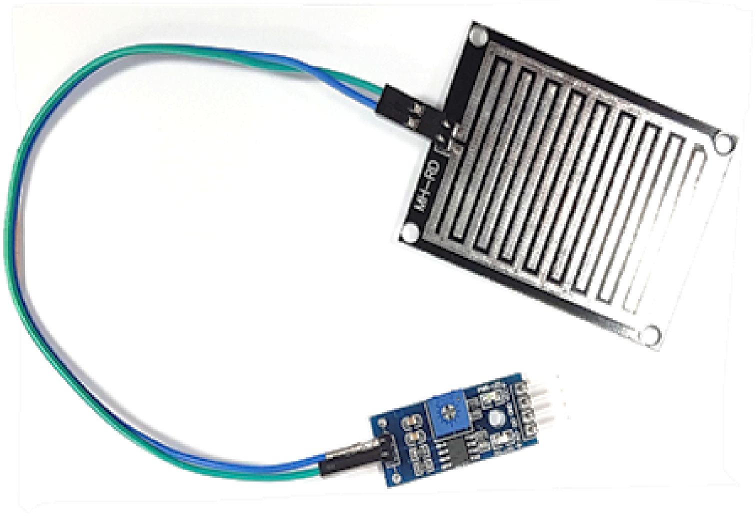


Figure 6.1: Rain sensor.

In Table. 6.1, we show the information related to the data gathered and sent by each sensor. In this table, we include the maximum and minimum possible values. The sensors may gather values higher than the maximum and lower than the minimum, but those values are not expected in the deployed areas. Moreover, we include in the table the sensitivity required from the sensor. The sensors could have higher sensitivity but for our application we require less sensitivity. The objective is to minimize the bits used to send data from each sensed parameter.

6.2.3 Architecture

The architecture proposed is a combination of nodes that measure data and a network infrastructure. This infrastructure not only has to be able to transmit the data, but also to provide a way to act intelligently and efficiently. Therefore, the network architecture chosen is Software Defined Network. That network builds the core of the system in terms of communication. It also can be connected to the Internet to use external services.

For improving the simplicity of the figures, the actuators that can appear together with the sensors in their networks are omitted. When a sensor appears in the figure, it can be both a sensor and an actuator. Fig. 6.6 shows the

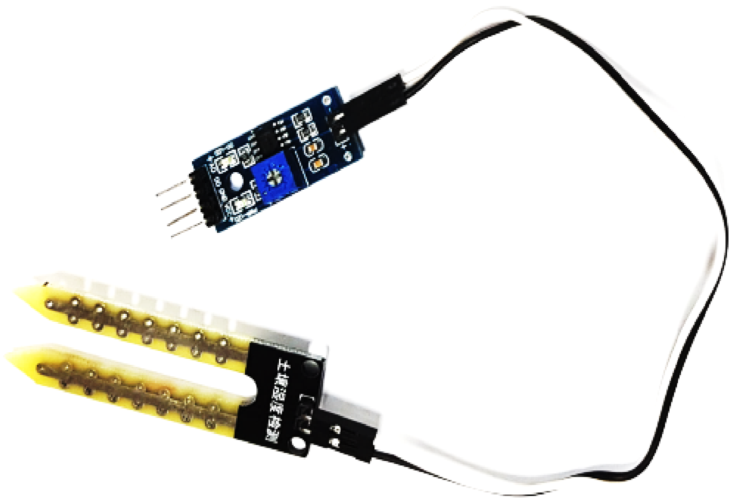


Figure 6.2: Soil moisture sensor.

picture of all the different actors in the architecture and their connections. On the one hand, the data gathering actors are connected to the core network. These actors can be either Smart Meters or sensor nodes. Furthermore, the sensor nodes can be meteo nodes (Weather stations) or irrigation nodes. The core network is an SDN composed by OpenFlow-enabled switches and access points (AP). On the other hand, the SDN can be also connected to the Internet. That is the data path of the system. However, an SDN controller is needed to manage the SDN and it is connected to every node in the SDN. The SDN controller is the one explained in chapter 3. In the figure, the connections used for managing the SDN are painted in brown.

This architecture is deployed in the Smart City, making several data source networks. These networks can be composed either by a Smart Meter or by a Wireless Sensor Network (WSN). WSN are made up of sensors gathering different data such as humidity and temperature. In addition, there are some actuators like garden hose for irrigation. Depending on the kind of source network, the gateway that interconnects the SDN with that source network varies. If the source network is a Smart Meter, the gateway will be an OpenFlow switch. Otherwise, the WSN need an access point because they are wireless. The access point is the cluster head(CH) of the WSN as well. These two dif-

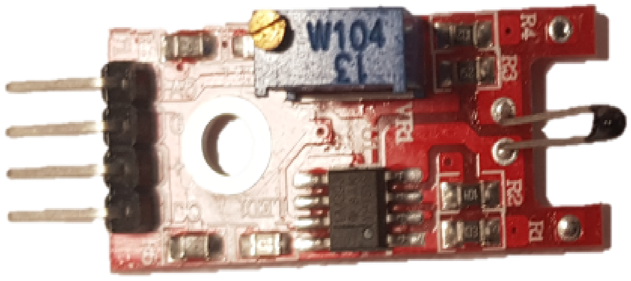


Figure 6.3: Temperature sensor.

ferent source networks are displayed in Fig. 6.7, when an example of multiple networks connected in the Smart City using the architecture is presented.

Finally, the layer architecture is showed in Fig. 6.8. The nodes in the architecture are classified into different categories or layers depending to their function. The first layer contains the nodes whose goal is to gather the data. The data is propagated through the network nodes, which belongs to the network layer. The data gathered by the smart meters and the sensors are sent to the SDN controller with the network statistics gathered by the network nodes. The controller provides management functions for the data and for the network, being able to command new measurements or data sending if the data is not well received. Furthermore, the controller is able to change network paths or processes if the network conditions change drastically. Nevertheless, the decisions needed to be taken are decided by the intelligence layer. The AI module, a software module that resides on the SDN controller, executes rules to decide what to do attending to the data gathered. The decisions are transformed into commands that are propagated down-way in the layer architecture. The actuators will execute the decided action like save water or irrigate. The sensors and the smart meters can act too, changing the measurement frequency or executing other functions.

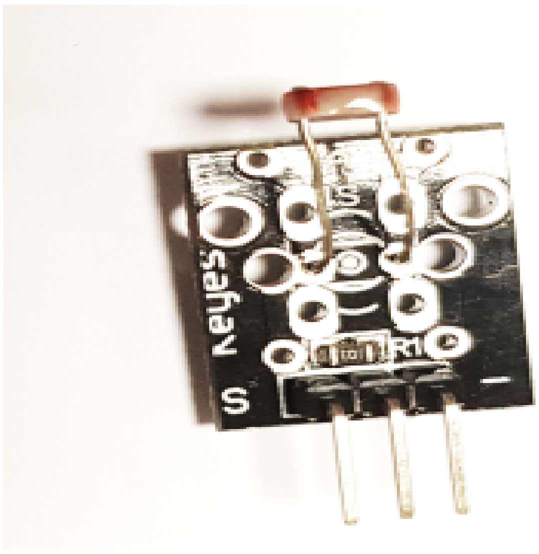


Figure 6.4: Solar radiation sensor.

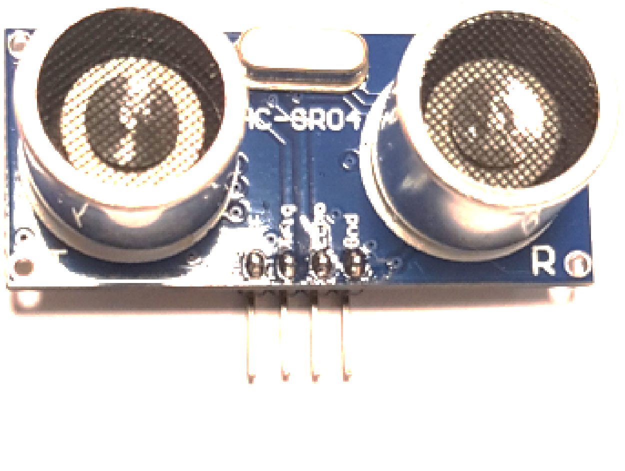


Figure 6.5: Presence sensor.

Table 6.1: Possible values gathered by each sensor.

Parameter	Min.	Max.	Interval	Sensi-tivity	Possible values	Bits
Presence (yes/no)	0	1	2	1	1	1
Moisture (%)	1	100	100	3.5	28.6	32
Water (yes/no)	0	1	2	1	1	1
Temperature (ºC)	-19	40	60	2	30	32
Rain (yes/no)	0	1	2	1	1	1
Wind (km/h)	0	31	32	2	16	16
Light (lux)	0	99,999	100.000	2000	50	64

6.2.4 Messages

Once the architecture and the aim of the system have been detailed, the next step is to define the messages used to provide the functionality to the system. The OpenFlow standard is used by the controller for gathering data from the network nodes. For example, when a sensor sends data to the CH, an OpenFlow message is used to forward the packet to the controller through the secure channel that connects the controller to the network nodes. However, some custom messages are needed to implement all the functionality of the system. Moreover, besides the messages used to communicate the data gathered by the sensors, other messages are needed to implement functions. The custom messages defined to be used by the system are the followings:

- ACK message: used to inform that the procedure carried out has been properly executed.
- Data: used to communicate the data collected from the sensors to the CH. This message is sent from any sensor or smart meter to the CH.
- Data Request: used by the SDN controller to request data to the sensors in a specific time instant. The sensors will perform a new measurement and will respond with a data message.
- Sleep: used to put a specific sensor or actuator to sleep during a specific period of time. It is sent from the SDN controller to any sensor or smart meter.

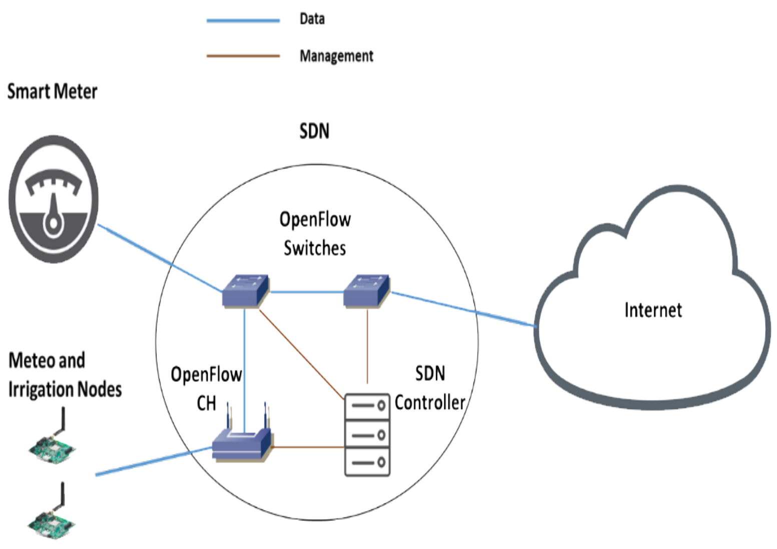


Figure 6.6: Architecture of the proposed system.

- Activation message: used to wake a sleeping sensor or actuator immediately. The SDN sends this message to a sleeping sensor or actuator.
- Act Time message: used to specify the waking time of a sleeping sensor or actuator. Like the activation message, it is sent from the SDN controller to a sleeping sensor or actuator.
- Stop irrigation message: used by the sensors to make the irrigation stop. This message is sent from the sensor to the actuators.
- Error message: used to inform that some error has appeared during the procedure.

All these messages are sent encapsulated in UDP datagrams. They include a header where the type of message and the length of the payload are detailed. Then, the data is presented. This structure is displayed in Fig. 6.9 where the messages are presented. Not all the eight messages appear in the figure due to some of them are empty messages. They are only composed by the type and length header and the length header value is zero. Those messages are: ACK message, activation message, stop irrigation message and error message. All of them are direct commands or info messages. The other messages need

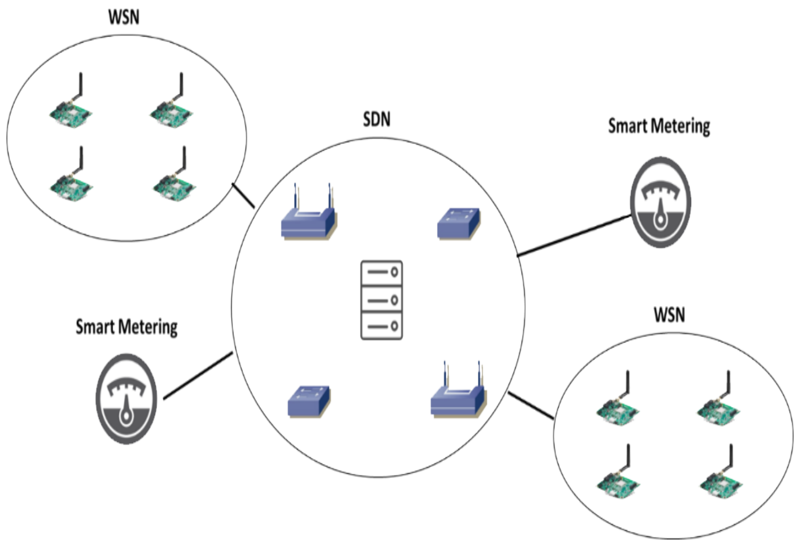


Figure 6.7: Example of multiple networks connected in the Smart City.

some fields in the value header to communicate data. Their fields are detailed in Fig. 6.9 among the type, length and value headers.

Accordingly, the system is constantly using OpenFlow standard messages to collect statistics from the SDN nodes and to forward messages to the controller but also the previously defined messages to execute actions and communicate the data from the sensors and smart meters.

As it is previously commented, the messages are sent through a UDP datagram. Therefore, the total size of the packets, either sent from or received by the sensors or actuators, is 79 bytes from the datagram, 3 bits from the type header, 1 byte from the length header and the length of the value header (6.1).

$$Size = Size_{UDP} + Size_{Headers} \tag{6.1}$$

Where

$$Size_{UDP} = 79 \quad Bytes$$

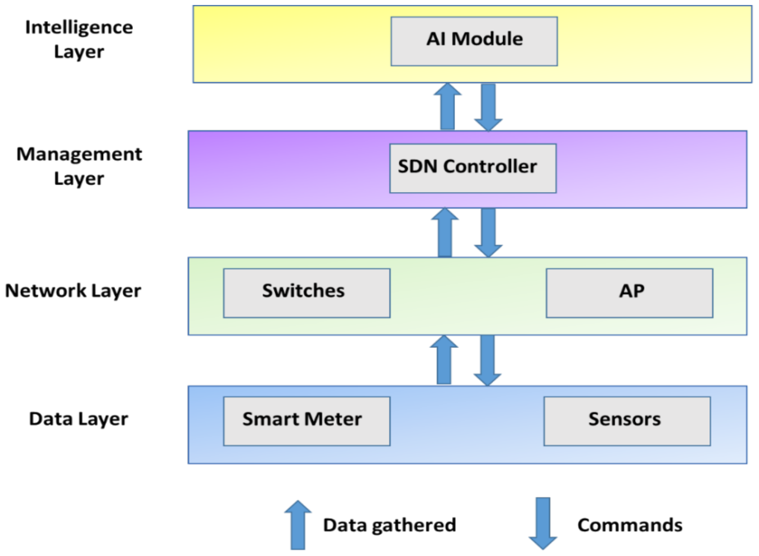


Figure 6.8: Layer Architecture.

$$Size_{Headers} = Size_{Type} + Size_{Length} + Size_{Value} \tag{6.2}$$

$$Size_{Type} = 4 \quad bits$$

$$Size_{Length} = 1 \quad byte$$

$$Size_{Value} = Length$$

The length of the packet sent depends on the type of packet. The ones that do not appear in Fig. 6.9 will have a size of 643 bits. However, data request message, sleep message and act time message will have a size of 651 bits. The size of the Data message will depend on the type of node. We manage four different kind of nodes, described in Table. 6.2, where the data sent by each kind of node is detailed. Each node sends a different amount of data and they are specified in the system just as in Table. 6.2. Therefore, the system is able to identify which kind of data is receiving and how many bits should read depending on the length header. This allows us not to use a “Node type” or “Data Type” field in the Data message, reducing the size of the message and the energy consumption. The Data message size will depend on the node,

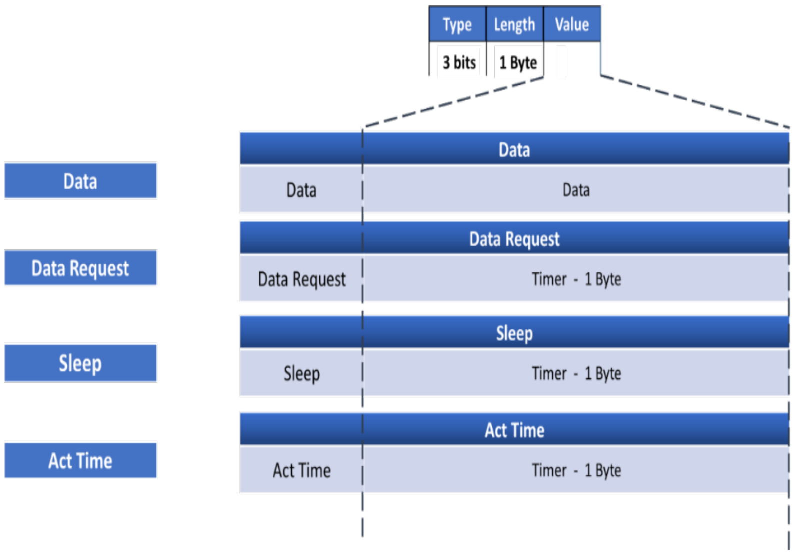


Figure 6.9: Messages with non-empty value field and their structure and length.

being 757 bits for a weather station node, 677 bits for an irrigation station node and 647 bits for a smart meter. In Table. 6.3 each kind of data type gathered by a different sensor is defined and its length is detailed.

With the messages presented, not only the communication process has been discussed, but also the size of the messages. The next point to discuss is the algorithm of the system.

Table 6.2: Data quantity and type sent by each kind of node.

Node Type	Data Sent by the Node	Data Length
Weather station node	Temperature, Rainfall, Wind, Light	114 bits
Irrigation station node	Humidity, Water Flow	34 bits
Smart meter	Consumption	4 bits

Table 6.3: Data length of each data type.

Data Type	Length
Temperature	32 bits
Rainfall	2 bits
Wind	16 bits
Light	64 bits
Humidity	32 bits
Water Flow	2 bits
Presence	2 bits
Consumption	4 bits

6.2.5 Algorithm

The irrigation process consists of a periodic evaluation of the scenario. Usually, every hour, the sensors and smart meters send the data to the SDN controller. The algorithm evaluates then the conditions for irrigating. The following factors need to be taken account before irrigating:

- If it is raining, the irrigation cannot start. The system should wait until the rain stops in future iterations.
- If the forecasting service indicates that will rain at the irrigation time, the irrigation is postponed.
- If there are people in the irrigation area (in those where people can walk on the grass), the irrigation does not start.
- If there are gust of wind greater than 20km/h, the irrigation is canceled.
- If the forecasting service indicates that there will be freeze, the irrigation is canceled.
- If the forecasting service indicates that there will be gust of wind, the irrigation is immediately started.
- If none of these conditions are true, the irrigation will start only if it is the irrigation time. This time is a period between the consumption peaks of the households at night and in the morning. The irrigation must start after the night peak and before the morning peak. Furthermore, if the area has an opening and closing time, the irrigation will be performed only after the closing time and before the opening time.

The flowchart of that irrigation process is showed in Fig. 6.10. First, all the data needed to be read is obtained. The sensor data, either obtained each hour from the sensor or consulted by the controller, the forecasting consulted in the web service and the consumption data gathered from the smart meters. Then, with all these data, the conditions are evaluated, and the decision is taken. This process is repeated every day several times until the irrigation is performed.

Some details of the process must be discussed. On the one hand, the irrigation will stop when the humidity sensor measures a humidity value greater than a defined threshold. This is performed by using the “Stop irrigation message” defined previously in the messages section.

On the other hand, the data of the consumption are processed by the AI module. Using a statistic model, the future peaks of consumption are predicted. Moreover, the changes of this consumption habit are also predicted. For example, if the consumption peak is delayed from 9PM to 10PM in summer, the system will adapt to that change. Therefore, the irrigation period will change dynamically depending on the users and water consumption problems during the irrigation process will be avoided.

However, prediction of consumption periods is not the only thing that the AI module can provide to the system. Some issues and problems like leaks or frauds can be also detected by analyzing the consumption histogram.

6.2.6 Requirements for the implantation of the proposal

In this section, we are going to present the network requirements if we try to implant the aforementioned proposal in urban environments. We are going to show the use of the proposal in different types of cities. First, we are going to describe 5 different scenarios.

- The first scenario is a small village with 100 houses, 3 business and few green areas >1000m2. Its green areas are mainly composed of roundabouts and small plots with grass and trees.
- The second scenario corresponds to a middle size town with 5,000 houses, 100 businesses and 20,000m2 of green areas. The green areas are composed by roundabouts, municipal gardens and a sport area.
- The third scenario is another middle size town with 5,000 houses, 100 business as the previous case bit with bigger green areas. A total of

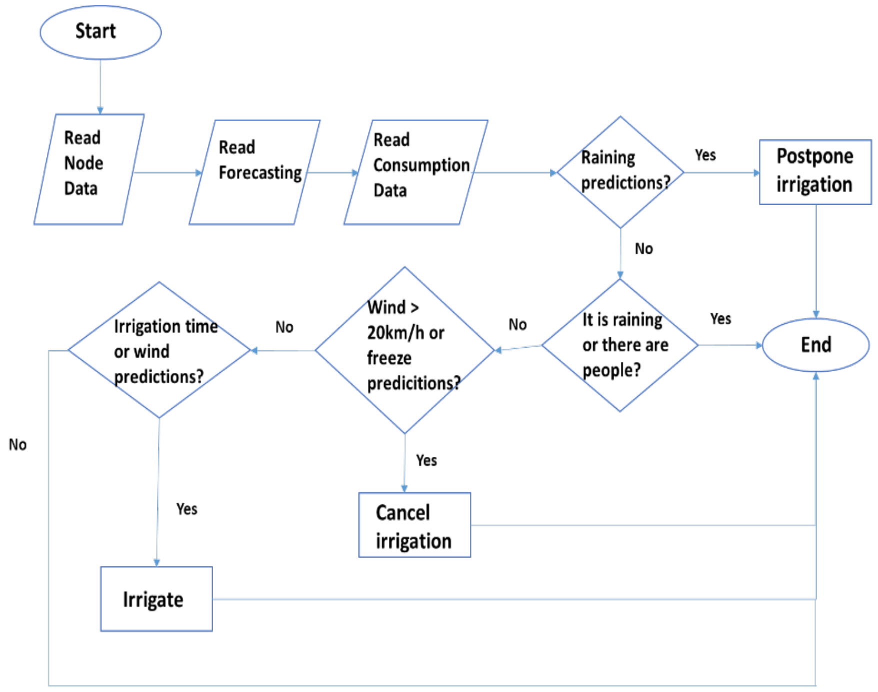


Figure 6.10: Algorithm of the irrigation process.

700,000m2 of green area are spread around the town. In addition to the aforementioned green areas in this town there is a golf park.

- The fourth scenario is a city with 200,000 houses, 25,000 business and 500,000m2 of green areas.
- The fifth example is a big city with 1,500,000 houses, 400,000 business and 62,000,000m2 of green areas. The green areas include different sport areas, roundabouts and big gardens and recreational areas.

Now, we are going to show the requirements if we want to deploy the proposed system in those scenarios. Thus, we are going to do some assumptions. The first one is to assume that in all the houses and business there is a smart meter installed. Next, the meteorological stations, where the temperature, rain, wind and solar radiation sensors are located, are placed in the green areas each 10.000m2. In the green areas, the humidity and the current sensors are placed at each sprinkler. This is called irrigation control station. We can assume that each sprinkler covers an area of 37m2 approximately. The sprinklers have

Table 6.4: Number of nodes for different purposes in each scenario.

City	**Smart Meters**	**Meteorological Stations**	**Irrigation Control Stations**
1	103	1	28
2	5100	2	541
3	5100	70	18919
4	225000	50	13514
5	1900000	6200	1675676

an overlapping of 25%. Thus, we can easily calculate the number of smart meters and stations located in each one of the aforementioned examples, see Table. 6.4.

Once we set the number of smart meters, meteorological stations and irrigation control stations we can calculate the amount of traffic generated in each case. The meteorological stations send messages of 114 bits of data and 643bits of headers. The irrigation stations send messages of 34 bits of data and 643 bits of headers. Finally, the smart meters send messages of 2 bits of data and 643 bits of headers. Thus, we can calculate the generated traffic when the SDN request data to all the nodes in each one of the aforementioned examples. The data obtained can be seen in the Fig. 6.11. In the example with the smallest town the data sent are 86.35Kbits and, in the biggest city, the data are 2368.43Mbits. We can see that the lowest traffic comes from the meteorological stations in all the cases. In four cases the major part from the traffic is related to the smart meters. Only in the third example, when a big surface is covered by green area, the data generated by the irrigation stations are the biggest part of the generated traffic.

6.2.7 Conclusions

In this section we have shown a smart irrigation system for urban lawns. The system uses data from different sensors placed at homes, gardens and water tubes. The objective of this work is to create a smart system able to decide the best moment to start the irrigation according to a series of given rules. Smart meters are used to monitor the water demand at homes in order to start the irrigation when the system detects that there is no water demand at homes. Several meteorological stations can be used to monitor the climatic conditions in order to delay the irrigation if it is raining or there is too much wind or too

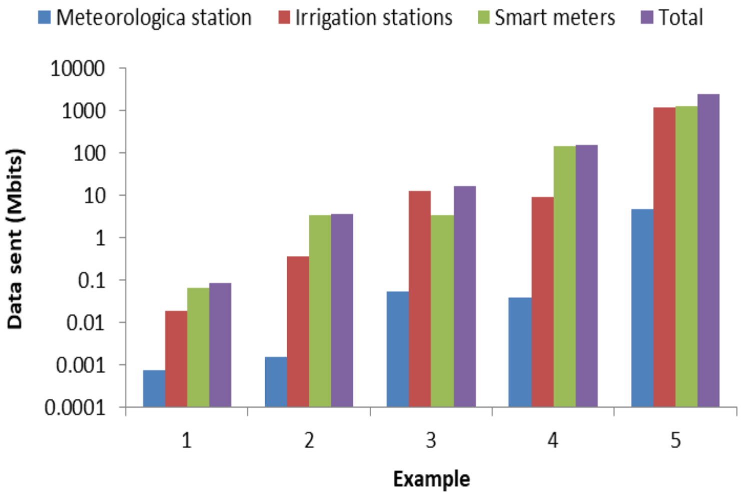


Figure 6.11: Data sent in each scenario to the SDN.

low temperatures. Thus, we can save water if we wait to irrigate if it is raining, avoiding misuse of water if there is wind or avoid damages in case of possible frost. Moreover, the system has sensors placed in the water tubes. These sensors are monitoring water flow and humidity in the sprinklers. The sensors in the tubes are placed to detect breakages and the soil moisture sensors close to the sprinklers monitor the water presence in the soil and determine when it is necessary to irrigate and when stop it. Using all these data and a smart algorithm the system expects to save some of the water used for irrigation and to reduce the water demand peaks. We have demonstrated the network bandwidth required in different scenarios when this system is used. SDN is used in this system to ensure the future scalability, the data security, and the inclusion of the smart algorithm in the network.

6.3 SDN for Video Surveillance

In this section, an intelligent system to manage multimedia transmission in a video surveillance IoT network is proposed. First, the architecture is shown. Then, the algorithm is described. Finally, the messages are displayed, and the communication process is commented.

6.3.1 Architecture

In this subsection, the architecture of the network is described. The system is designed for surveillance. The architecture of the proposal is a combination of two network technologies. This combination is shown in Fig. 6.12. On the one hand, IoT networks work as edge networks. The IoT nodes implement the functionality of the system. On the other hand, an SDN is used to provide the core network. The SDN controller, whose function is to ensure the best QoS, is the central node in the network and it is able to make decisions to interconnect the different IoT networks.

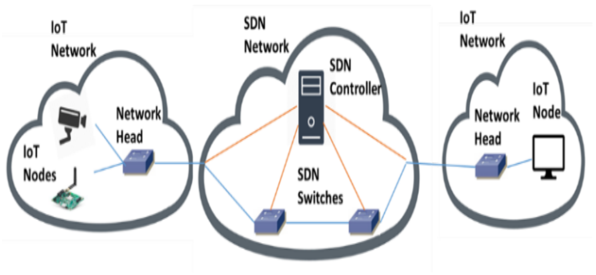


Figure 6.12: Scheme of the architecture proposed.

The two networks are joined by the Network Head (NH) of each IoT network. It is a special node that manages the IoT network communication and sends the data through the SDN network. Moreover, it uses the OpenFlow standard to communicate with the controller and send it statistics about the use of the network. This role is played by an OpenFlow-enable switch. There are also OpenFlow-enabled switches that do not act as the NH of an IoT network. However, the SDN controller has the AI module and it is in constant communication with it. The AI module is a set of software programs that uses AI techniques to provide the functionalities to the system proposed.

Thereby, the network is composed by IoT networks that implement the functions. There are several different kinds of IoT nodes in the system. Fig. 6.13 shows the different roles in the system and the communication between them. The different tasks that each role implements are painted in different colors. The SDN controller is in charge of network management and sends the statistics that gather from the SDN nodes (NHs and other SDN switches) to the AI module. The AI module uses this set of data in order to apply the AI techniques and inform the controller about the multimedia traffic flowing through the network and its resource requirements. This module is divided into two parts: The traffic classifier, which reports whether the incoming flow is critical or not; and the estimator, which decides the kind of action that should be performed by the SDN controller in order to guarantee the QoS conditions for multimedia transmission.

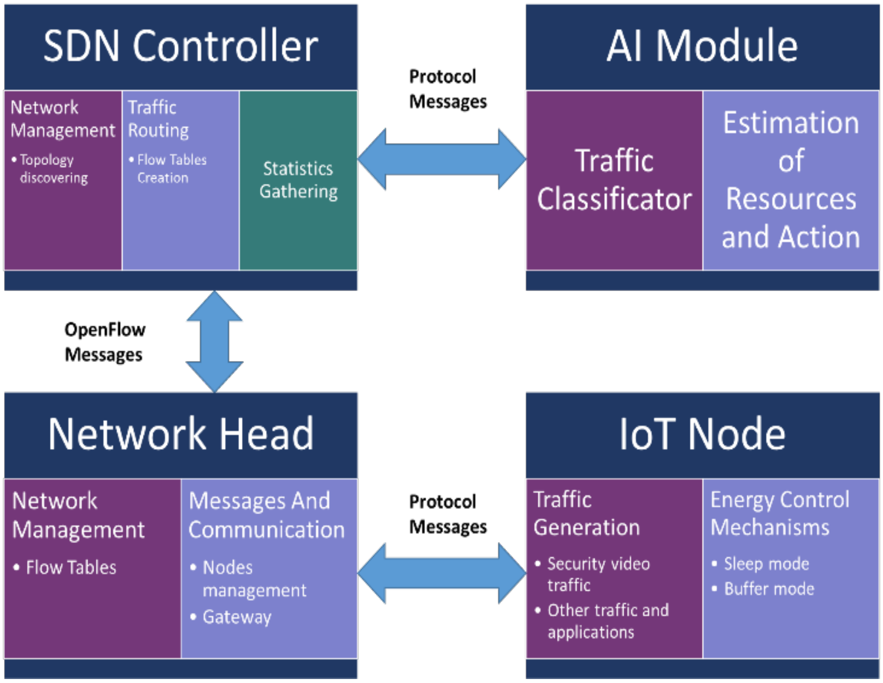


Figure 6.13: Scheme of each actor in the system and their interaction.

The communication between the SDN controller and the AI module is internal, but it is composed by structured messages. The SDN controller performs network management functions. However, the most important parts for this paper are the traffic routing part, where the SDN controller fills the flow table of each switch, and the statistics gathering. In order to do that, the controller communicates with the SDN nodes by using the OpenFlow standard. Nevertheless, in some cases we need to use custom messages defined in the following subsections. The Network Head not only performs network management tasks, but also communicates with the IoT nodes and it is able to manage their behavior. With the messages described in Subsection 6.3.3, the Network Head is able to operate with the nodes. These nodes are the ones that generate the traffic in the network. Some of this traffic is multimedia traffic, like video surveillance traffic, and its QoS must be guaranteed.

6.3.2 Algorithm

In this subsection, the algorithm performed by the SDN controller is detailed, and the different actions that can be executed either by the controller or the nodes are described and classified. The algorithm of the network management is simpler than the one used by the AI module. The controller initializes the AI module. Its main task is to use the standard OpenFlow messages to gather statistics. The algorithm used for network management uses the AI module to detect critical traffic flows being sent through the network. This critical traffic means multimedia traffic in this paper. When multimedia traffic is detected by the AI module, it estimates the resources required and the best action to perform in order to provide an acceptable level of QoE. This estimation uses the statistics provided by the controller to be aware of the current state of the network. With the estimation done by the AI module, the SDN controller chooses an action to perform in the network. Depending on the resources needed to provide enough QoE in the transmission, the AI module labels the level of urgency of the actions to be taken. Therefore, the SDN controller handles the categories of the actions. The SDN controller will perform an action categorized into the same level of needed resources (listed in Table. 6.5) as that of the AI module. With these actions, the SDN controller is able to change priority, queuing policies or making routing decisions in order to guarantee the QoS needed in the transmission. Some problems, such as media access, are managed by the controller. If the action being performed by the SDN belongs to the first three categories, the OpenFlow standard does not contain any messages that could perform it. Therefore, we have designed some custom messages (explained in the next subsection) that implement these actions. The action labeled as IoT has been specifically designed for the architecture maintenance. They allow activating backup nodes in destination and enable buffer mode in source. Moreover, with this ability to put some IoT nodes into sleep mode, the system will avoid QoS problems.

The algorithm is described in Algorithm 6.1 and it is graphically shown in Fig. 6.14, where the Execute_Action subprocess is detailed. The SDN controller starts gathering statistics and providing these statistics to the AI module. When a new packet is reported to the controller, it sends the packet to the AI module. The AI module detects if it is a critical situation. If it is, the AI module also estimates the action to perform. The controller is notified about the action and it sends the messages needed to perform this action in the SDN nodes. The messages needed to be sent change depending on the action to be executed. When the action is done, the controller returns to its usual stats reporting activity.

Table 6.5: Possible actions and their category.

Category	Action
Queuing	Change the queuing priority
Queuing	Use QoS queuing
VLAN	Use VLAN PCP for packet priority treating on each switch
BW	Enable an alternative route
BW	Enable load balancing
BW	Send the traffic through a priority treatment route
BW variation	Enable link aggregation
IoT	Back up nodes in destination
IoT	Enable buffer mode in source node
NH Congestion	Sleep mode in source IoT network

6.3.3 Messages

The messages sent from the controller to the nodes belong to the OpenFlow standard in almost all cases. In this subsection, the communication processes described are not only those between the SDN controller and the nodes, but also between the SDN controller and the AI module. In addition, the structure of the messages added in order to expand the capabilities of the SDN controller is also detailed.

Since the SDN controller contains the AI module, the messages exchanged between them are not sent through the network. However, it is very important to describe this communication in order to understand how both actors work. The communication process in which the SDN controller receives a new packet is shown in Fig. 6.15. The OpenFlow "Packet_In" message is sent from the NH to the controller. The controller sends the packet to the AI module, which classifies the new flow and decides if it is critical or not. Then, if the flow is critical, the AI module estimates the level of criticalness depending on the state of the network. These states have been built up thanks to the messages sent by the SDN controller to the AI module (displayed in Fig. 6.16. These messages contain the statistics gathered by the nodes. The AI module reports its classification to the SDN controller. With this info, the SDN controller executes an action that matches with the estimation performed by the AI module.

Algorithm 6.1

```
Given: Actions

Initialize_AI(Actions)

Cat_Prev = Cat_initial
Foreach new iteration
    Stats = Get_Statistics()
    AI_Sent_Statistics(Stats)
    If New_Packet
        Cat = AI_Send_Packet(Packet)
        If Cat != No_Crit
            Execute_Action(Cat)
        End If
    End If
End Foreach
```

Depending on the action to be executed, the OpenFlow messages will change. Table. 6.6 shows the actions taken when each message is sent.

A special case is when the action is related to the IoT networks. In that case, there are no OpenFlow standard messages to implement the actions, so the messages shown in Fig. 6.17 are used. They are the following ones:

- Category: Used to communicate with the AI Module and reports which kind of resource has to be improved.
- Sleep: Used to inform the NH that all the networks in the IoT network must be put to sleep when the timer reaches 0 except the video source.
- Awake: Used to inform the NH that all the networks in the IoT network must be awakened when the timer reaches 0. Used also to activate the backup nodes in the destination IoT network.
- Buffer_Mode: Activates the buffer in the video source indicated by its ID.

The communications steps differ when an OpenFlow message is used and when one of the custom messages are used. This happens because the custom messages are focused on IoT network management. The communication process of each case is shown in Fig. 6.18 and Fig. 6.19. This process is the continuation

Table 6.6: Actions taken when each message is sent.

Action	Messages Used
Change the queuing priority	OF_PACKET_OUT (ENQUEUE)
Send the traffic through a route with higher priority	OF_FLOW_MOD
Enable load balancing	OF_FLOW_MOD
Enable an alternative route	OF_FLOW_MOD
Enable link aggregation	OF_FLOW_MOD
Use VLAN PCP for high priority packets on each switch	OF_PACKET_OUT
Back up nodes in destination	Awake
Enable buffer mode in source node	Buffer_Mode
Sleep mode in source IoT network	Sleep

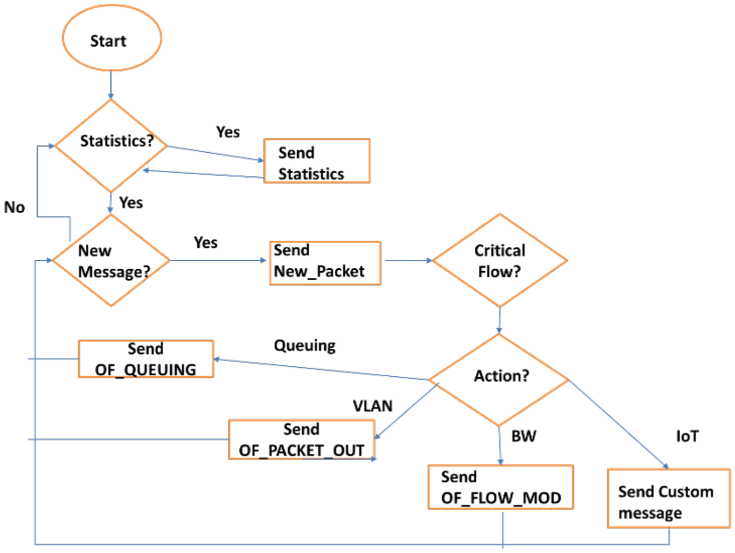


Figure 6.14: SDN controller operation diagram.

of the one described in Fig. 6.15 (when the AI module indicates that the flow is critical and the action that should be executed). Fig. 6.18 shows the message exchange when the action is focused on the SDN nodes. The SDN controller uses the OpenFlow messages to inform the SDN nodes of the network on which action must be performed. Fig. 6.19 describes the process when the IoT network is involved and the NH is the destination of the messages sent by the controller.

6.3.4 Methodology and Results

Once the AI system is trained and explained, its application to the SDN must be measured. In this section, the experiments performed to measure the improvement during multimedia transmission are detailed. First, the topology and the software used in the experiments are described. Then, the results are shown and discussed.

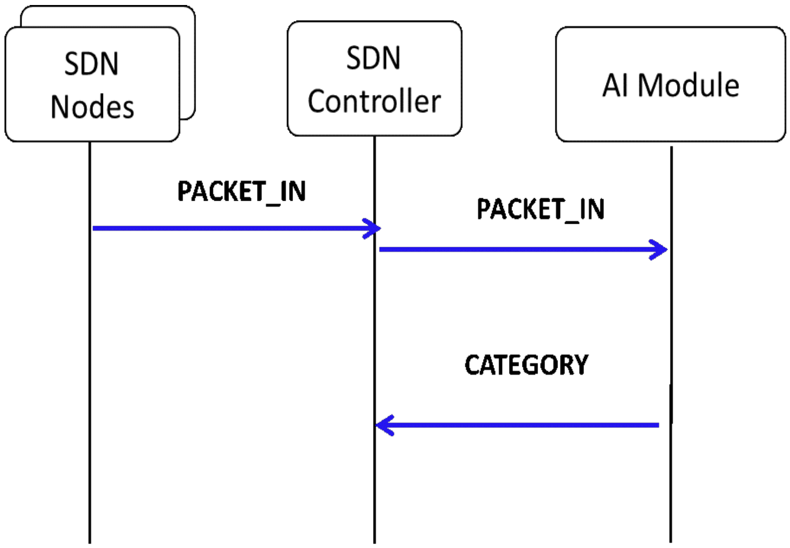


Figure 6.15: Message communication process when a new packet arrives.

Methodology and Topology

The experiments ran over the emulator Mininet. This emulator provides SDN emulation by using Linux Hosts as PC and Switches. The experiments consist of sending video streaming through the core network in different scenarios. The streaming is performed using the VLC software. Both the source and the destination network are emulated as Linux hosts in Mininet; so, we manage them as hosts in the network.

Mininet allows us to modify the network conditions and, along with the multimedia traffic sent, to simulate different scenarios. These conditions are defined in a script and the network is built with those characteristics. The topology used in the experiments is the one shown in Fig. 6.20. This topology allows us to use different techniques, such as alternative routing, thanks to the path redundancy. The streaming source is H1, and the destination, H2. The path chosen for the delivery is S1-S3-S5 because the others present higher delay. Moreover, the link between S1 and S3 is marked in Fig. 6.20. This is because we are able to use link aggregation in that link in order to increase the available bandwidth.

The different scenarios tested are the following ones: In the first scenario (scenario 1), there is a bandwidth problem and the action indicated by the AI

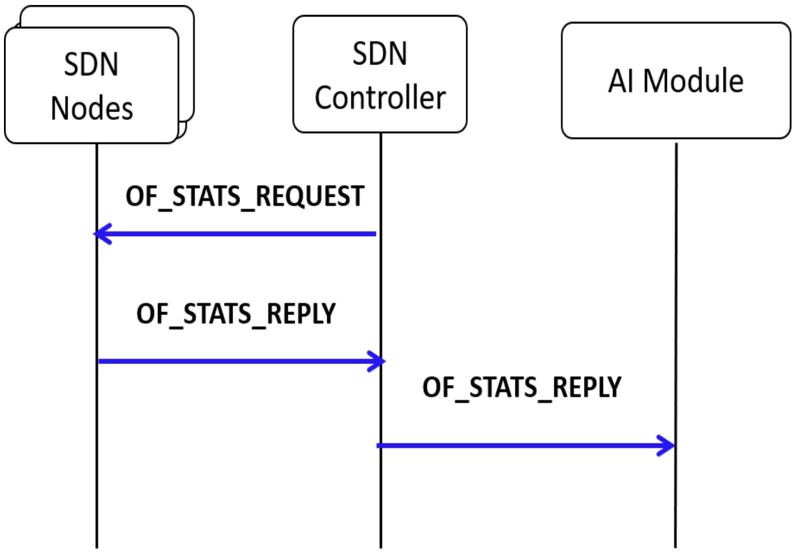


Figure 6.16: Message communication process when statistics are demanded.

module is to use an alternative path, with an increment of delay. In the second one, the problem is the same, but the action to perform is to use link aggregation in order to increase the bandwidth. In scenario 3, the AI module suggests using queuing to modify the priority and reduce the loss rate. Finally, in scenario 5, there is congestion in the source IoT network, and the AI module reports the custom message that is necessary to put the nodes to sleep. In the next subsection, the results of each scenario are presented.

Results

The results of the experiments are displayed and discussed in this subsection. Not only are QoS parameters measured, but also QoE has been analyzed, and it is discussed at the end of the subsection.

In Fig. 6.21, the comparison between the bandwidth used by the multimedia streaming in both cases, with and without the proposed system, is displayed. Without any system that performs actions to improve the QoS, the bandwidth has a maximum of 1.83Mb/s. However, with the proposal, that maximum increases up to 3.08Mb/s. The minimums are 16.4kb/s and 104.12kb/s, respectively. The average bandwidths are similar, 1.19Mb/s and 1.08Mb/s.

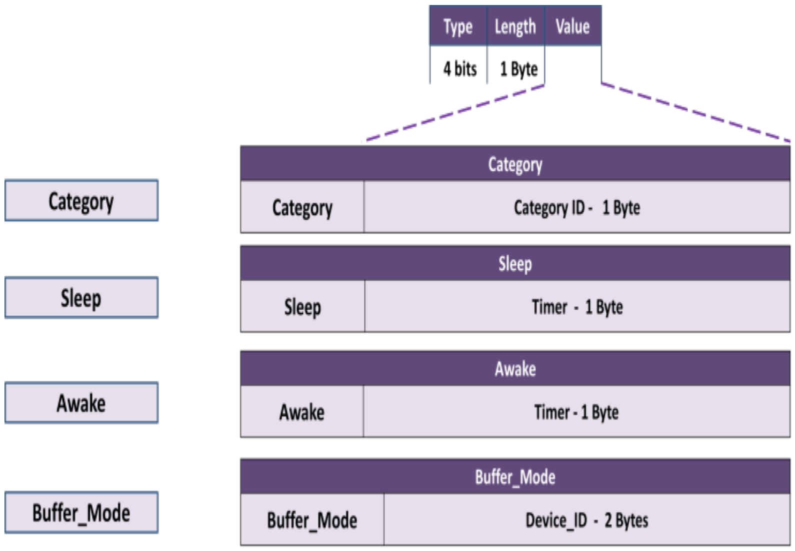


Figure 6.17: Custom messages used when actions in the IoT networks are required.

In terms of jitter, the performance is compared in Fig. 6.22. The average jitter without the proposed system is 2.47ms. However, by using the alternative path, the average is 8.19ms. The maximums are 11.13ms and 47.62ms, respectively. Finally, the minimum jitter also increases from 0.01ms to 0.13ms with the proposed action.

In scenario 2, due to the network status, the action to perform is link aggregation. Fig. 6.23 shows the changes produced in terms of bandwidth when the system is used. Without it, the average bandwidth is 1.12Mb/s. There was a maximum of 1.89Mb/s and a minimum of 10.9kb/s. When the system is used, the average bandwidth is 1.42Mb/s, the maximum is 2.99Mb/s, and the minimum is 16.4kb/s.

Regarding the jitter, there is a reduction when the system performs the action indicated by the AI module, as shown in Fig. 6.24. The average jitter is reduced from 2.4ms to 0.49ms. The maximums are similar: 8.89ms when the proposal is not being used, and 7.63ms when it is. The minimums are 0.22ms and 0.03ms, respectively.

In scenario 3, the loss rate is measured. The AI module decides to use priority techniques using queues. Fig. 6.25 shows that the bandwidth does not change like in the previous scenarios. The average bandwidths are 1.01Mb/s,

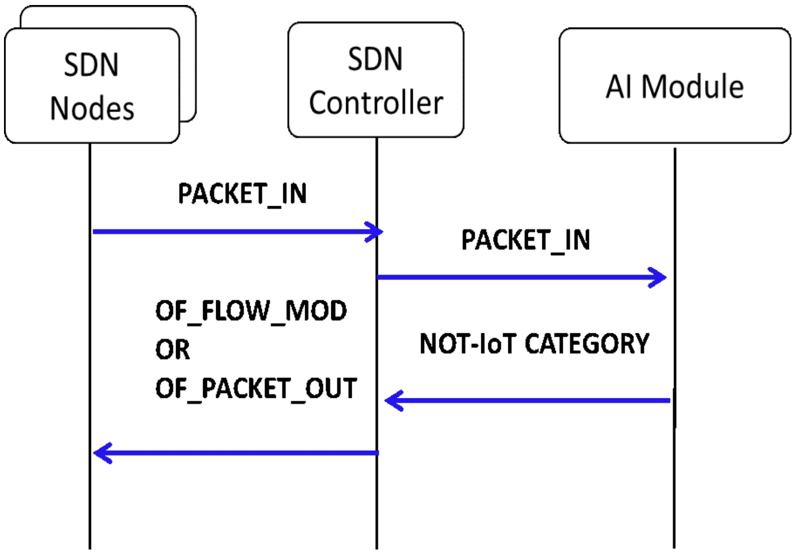


Figure 6.18: Message communication process when there is an SDN-node-based action.

without applying the action, and 1.07Mb/s when applying it. The maximum is 1.95Mb/s in both cases and the minimum is 16.4kb/s.

However, the jitter presents some differences. Fig. 6.26 shows that the average jitter without the proposal is 4.77ms, and it is reduced to 1.35ms when it is used. The average jitter is quite different: 10.2ms for the proposal, and 43.05ms when it is not used. The minimums are 0.01ms with the proposal and 0.13ms without it.

Fig. 6.27 shows the reduction of the loss rate from 9.07% to nearly 1.16%. This loss rate reduction works to improve the QoE, as is shown at the end of the subsection.

The last scenario is used to test the custom characteristic for QoS improvement using IoT characteristics. The congestion in the source network is handled by putting the rest of the nodes to sleep. So, the QoS of the multimedia transmission is improved, as shown in Fig. 6.28. It shows that the maximum bandwidth consumed by the multimedia flow is increased from 1.79Mb/s to 2.92Mb/s by using the proposed solution. The average bandwidth is also increased from 0.85Mb/s to 1.24Mb/s. The minimum bandwidth consumed is 8.98kb/s with the system and 104.1kb/s without it.

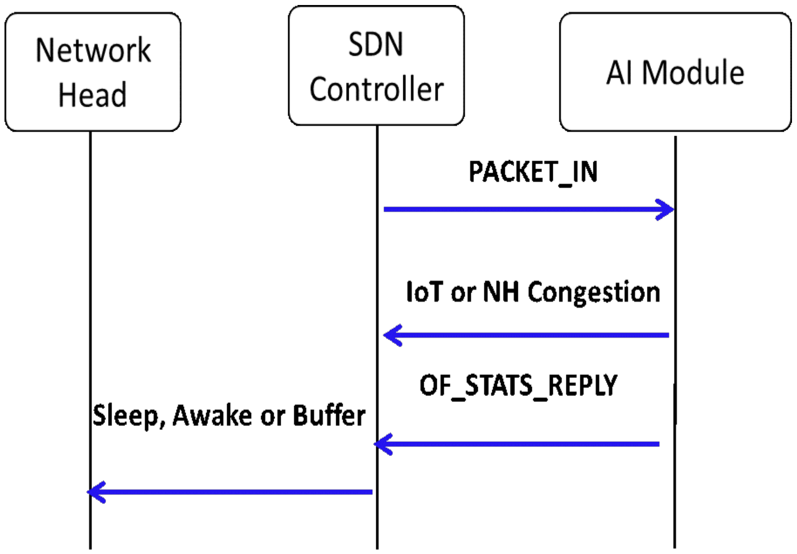


Figure 6.19: Message communication process when the IoT network is affected.

The jitter also changes with this action. It is displayed in Fig. 6.29. The congestion in the source network produces an average jitter of 9.93ms. There is a maximum of 48.08ms and a minimum of 0.12ms. Nevertheless, with the action performed, this jitter is reduced to 0.27ms of the average, 6.19ms of the maximum and 0.01 of the minimum.

After testing these scenarios, the video obtained in the destination network is watched by 11 users, 8 males and 3 females. They chose for each player a score from 1 to 5, 5 being the best quality and 1 being the worst one. The results are shown in Fig. 6.30. All the actions performed by the controller increase the MOS obtained by the users. On the one hand, the greatest improvements are those produced in scenario 3, where the MOS increases from 2.1 to 5, and in scenario 4, where it increases from 1 to 3.2. On the other hand, in scenario 2, the increment is from 3 to 3.75 and, in scenario 2, the MOS increases from 3.2 to 4.1.

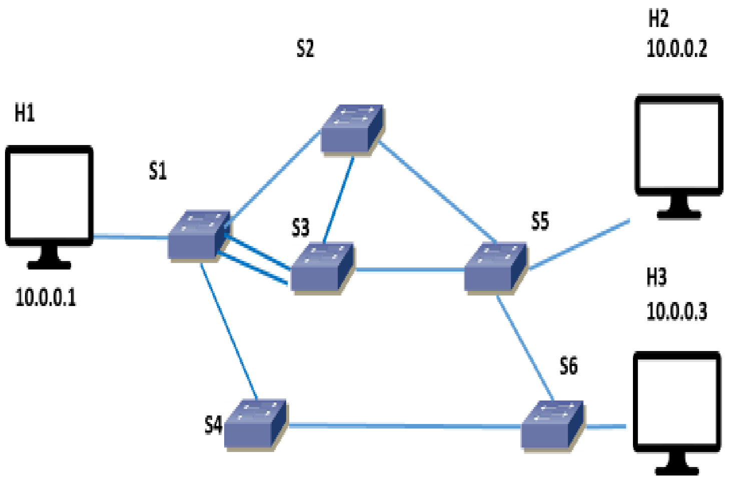


Figure 6.20: Topology used in the experiments.

6.4 SDN for Emergency Managemenet in Smart Cities

This section presents an SDN implemented in a smart city that provides a management system in emergency situations, notified by alarms. First, a typical scenario and the IoT architecture proposed in this work are shown. The section also includes the packet exchange for both the case when an alarm is detected and the case when the alarm is finished. Finally, the employed structure of the messages is described.

6.4.1 Architecture and Algorithm

IoT networks are composed by nodes that provide data to the network in order to make intelligent and efficient decisions. On the other hand, an SDN is composed by virtual devices, specifically Openflow Switches, which play the role of the CH. In our scenario (See Fig. 6.31), the traffic elements such as cameras and traffic lights compose each IoT network while each OpenFlow switch is managed by an SDN controller that can make decisions about the behavior of the nodes on the IoT networks. The SDN network is in charge of gathering data from all IoT networks and distributes this information along the rest of the OpenFlow switches. Additionally, the SDN controller communicates

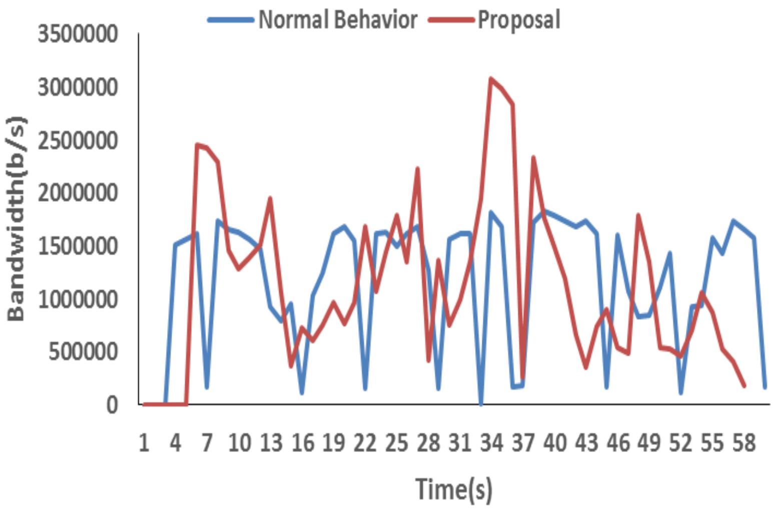


Figure 6.21: Bandwidth obtained in scenario 1.

with a cloud service which is capable of requiring actions from the controller in order to make changes in the SDN network and in the IoT networks.

As Fig. 6.31 shows, the entire city is a smart city where the control plane is directed by the SDN network and the data plane is composed by the IoT networks. When an emergency event is registered, the SDN controller receives data from the OpenFlow switches and the SDN controller sends requests for available resources throughout all the networks, as well as the location (by using area IDs) of each IoT network and the possibility of fast reaching the place where the emergency was detected. At that moment, the flows are opened to allow communication between the IoT nodes that have to collaborate in order to divert urban traffic and make decisions so as to obtain the fastest paths for the emergency services.

In order to ease the access of emergency services to the place where the emergency was detected, the SDN network modifies the behavior of the traffic lights changing, for example, the duration of the intervals between sequences of green / amber / red lights for vehicles, the time pedestrians spend crossing or the diversion of vehicles to certain directions. Traffic lights and traffic panels can be employed to modify the routes for vehicles and disperse them from the place of the emergency. In this way, it is possible to coordinate traffic lights that initially were not and, after varying the routes or conditions, this cooperation is

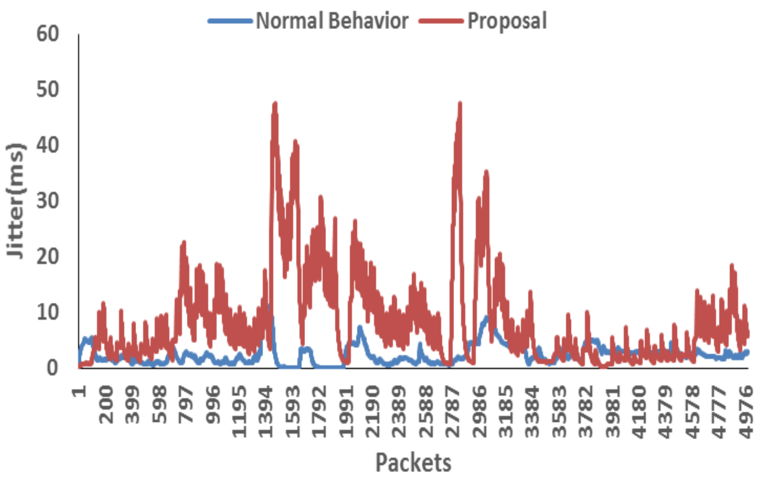


Figure 6.22: Jitter obtained in scenario 1.

needed for better management of the emergency situation. The pseudocode of the algorithm is presented in Algorithm 6.2. First, the type of the emergency is obtained from the emergency ID, if the emergency is labeled as a high-risk emergency, additional security resources are asked. Then, the shortest route is calculated for each resource, and taking into account that some resources can share parts of the route to the emergency area. Finally, the SDN flows of the CH are modified, the CH is informed about the emergency and the behavior of the IoT nodes are communicated.

In order to completely understand the algorithm, the messages exchanged are described in the next subsection.

6.4.2 Messages

In this subsection, the structure of the exchanged messages is detailed. The exchange of messages when an alert occurs is displayed in Fig. 6.32. Fig. 6.33 represents the flow of the messages when the emergency has been handled.

Fig. 6.32 describes the process of communication when the emergency service notifies an emergency alert. It sends the message to the SDN controller that read the data to know if the emergency has been classified as a high-risk

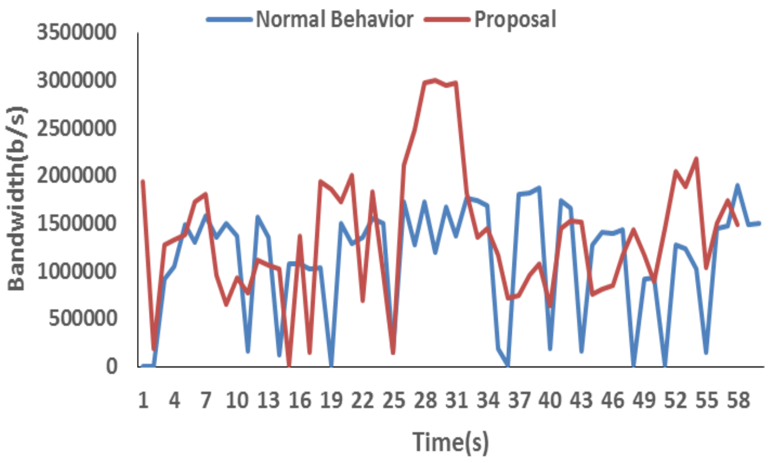


Figure 6.23: Bandwidth obtained in scenario 2.

emergency. If it is a high-risk emergency, the SDN controller asks for resources to the local security service located in the city. The service provides local security resources and notifies there are via Resource Allocation messages. After that, the algorithm determines the routes for the normal traffic and for every emergency resource available. By using OpenFlow Flow Modification messages the controller changes the communication between the IoT networks. Then, it informs the CH of the affected areas that the behavior of the node has to change due to an emergency, using the FL Emergency Behavior message. The FL Behavior set the behavior of an individual IoT node.

In a similar way, Fig. 6.33 describes the communication established to set the behavior of the nodes back to the usual way of operating. The message flow is similar to the described in Fig. 6.32. Instead of asking for resources, the SDN controller informs that the resources have been released because of the end of the emergency. Then, modifies the flow table again in every CH and informs them that they have to change the behavior of their nodes to the one that has been previously used. That is performed via FL Normal Behavior message.

The structure of the messages is used in the next subsection for calculating the energy consumed in the transmission during an operation.

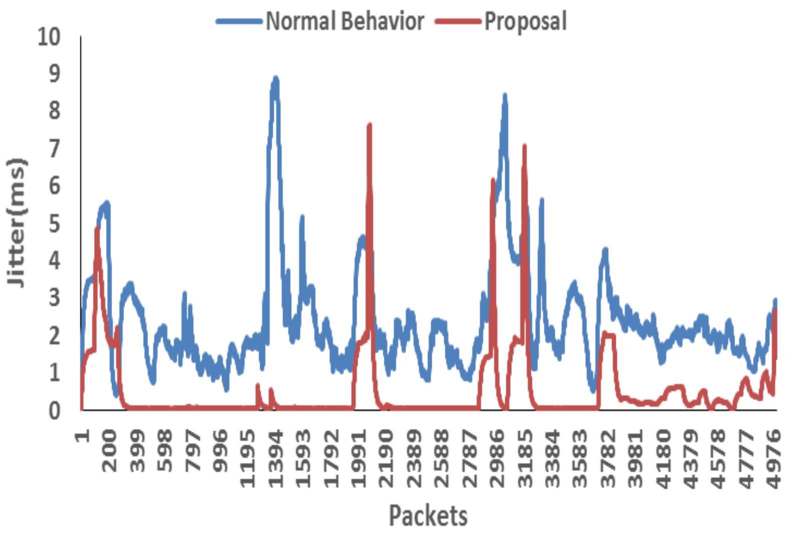


Figure 6.24: Jitter obtained in scenario 2.

In Fig. 6.34 the messages using in the algorithm are described. All the messages used in this proposal follow the structure TLV (type, length, value). As it is showed in Fig. 6.34, the field Type is composed by 4 bits, and can describe 16 different types of messages. Then, the field Length (1 Byte) is used to inform about the size of the message. Finally, the field Value contains the data that the applications need in order to work properly.

The messages used in the protocol are:

- FL Behavior: Assigns a concrete behavior to a traffic control node.
- FL Normal Behavior: Informs the CH that it must configure its nodes back to the normal behavior.
- FL Emergency Behavior: Notifies the CH that an emergency has been alerted and the controller will send FL Behavior messages to modify the workings of the nodes.
- Release of Resources: It is utilized to advise the local security service that the resources allocated to an emergency are not used anymore.
- Ask for Resources: Requests the allocation of local security resources to an emergency.

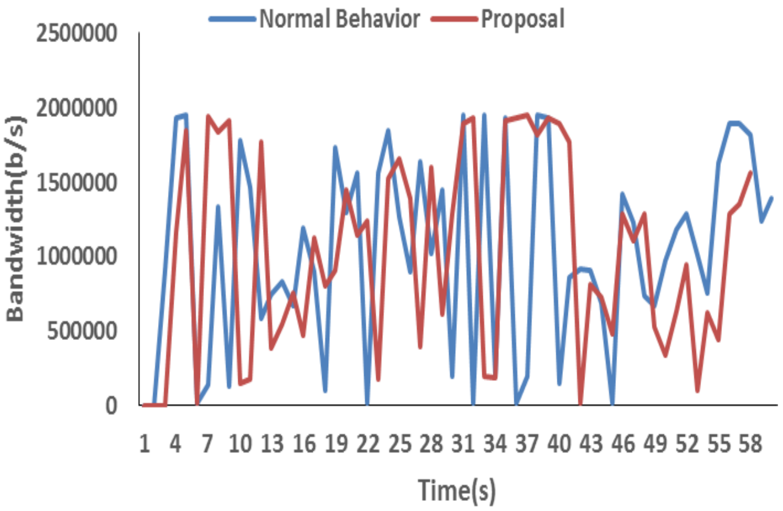


Figure 6.25: Bandwidth obtained in scenario 3.

- Resource Allocation: Informs that the resource is assigned to the emergency. As well as the area ID and the number of resources that the service is going to allocate then.
- Emergency Alert: Message used by the emergency system to inform that there is an emergency.
- Emergency Finished: The emergency system uses this message to notify that the emergency is finished, and the resources allocated to it have been released.

The value of the field Type indicates which of the previous messages is being sent. Then, the length field specifies the size of the Value field, where the important data is located. The ACK message has no value field and its length field it is equal to zero.

In order to invoke the local security unit service, the emergency has to be classified into high risk and low risk. This classification, which is done by the emergency service, it is notified by using the Emergency ID in the Emergency Alert message. The first bit of the field Emergency ID is set to 1 if the emergency is classified as high-risk emergency. Therefore, there are 256 emergency IDs, 128 low risk IDs and 128 high risk emergency IDs.

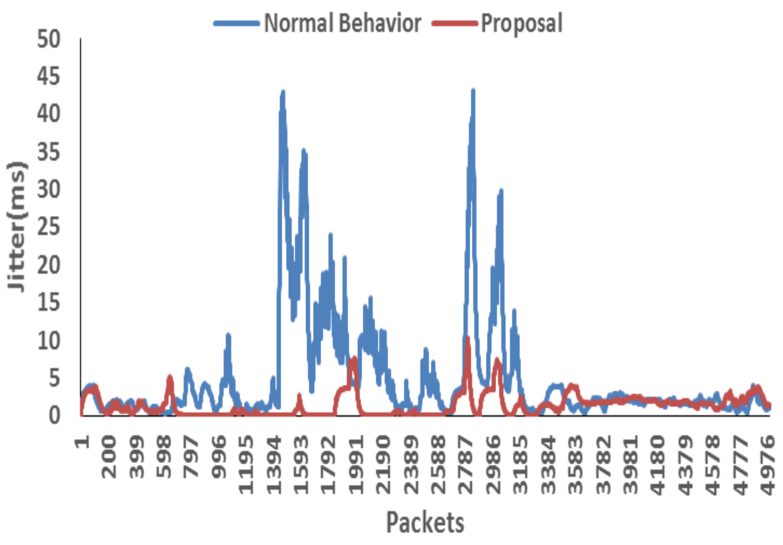


Figure 6.26: Jitter obtained in scenario 3.

The emergency ID is used to identify a concrete emergency during an established period of time. Both the controller and the emergency service can expand the emergency ID or generate a new one in their respective databases in order to manage them and enhance their applications or provide some new.

6.4.3 Energy Consumption Model

In this section, the energy consumption model used to measure the energy consumed in the entire emergency treatment process is described.

As is detailed on [197], the energy consumed per bit in the transmitter-receiver model can be modeled by the equations (6.3) to (6.7).

$$E_{bit} = E_{bit-tx} + E_{bit-out} + E_{bit-rx} + E_{bit-cod} \tag{6.3}$$

Where

$$E_{bit-tx} = \frac{P_{tx-e}(\frac{header+payload+trailer}{rate}) + P_{tx-ini}T_{tx-ini}}{payload} \tag{6.4}$$

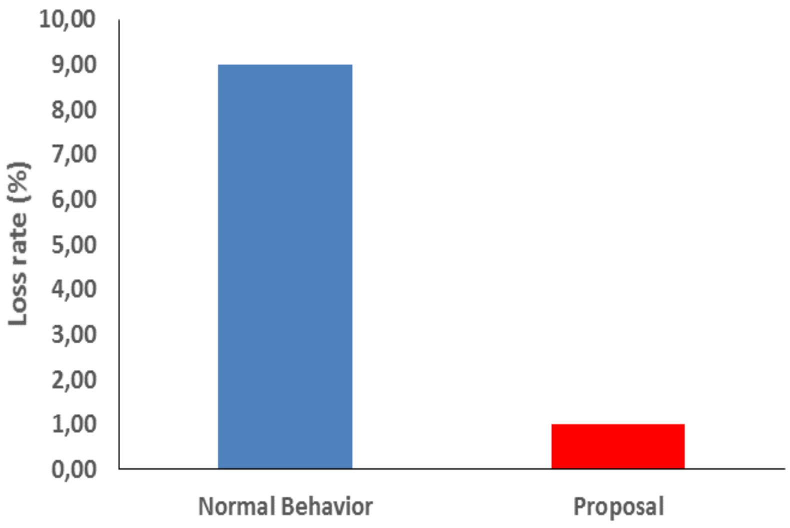


Figure 6.27: Loss rate obtained in scenario 3.

$$E_{bit-out} = \frac{P_{tx-out}(\frac{header+payload+trailer}{rate})}{payload} d^2 \tag{6.5}$$

$$E_{bit-rx} = \frac{P_{rx-r}(\frac{header+payload+trailer}{rate}) + P_{rx-ini}T_{rx-ini}}{payload} \tag{6.6}$$

$$E_{bit-cod} = \frac{E_{coding} + E_{decoding}}{payload} \tag{6.7}$$

and

E_{bit-tx} = energy consumed per bit in electronic transmission

$E_{bit-out}$ = energy consumed per bit in output transmission

E_{bit-rx} = energy consumed per bit in electronic reception

$E_{bit-cod}$ = energy consumed per bit in coding/decoding processes

It is assumed that the energy dissipated in transmission (P_{tx-e}) 50 nJ/bit, is exactly the same as the energy dissipated in electronic reception (P_{rx-e})

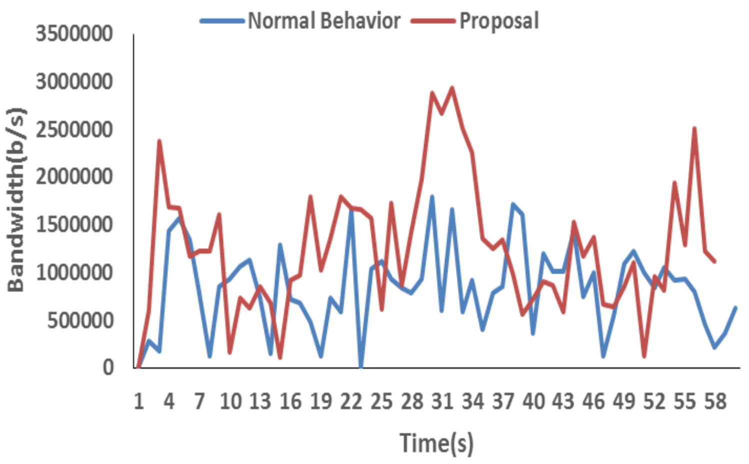


Figure 6.28: Bandwidth obtained in scenario 4.

the other hand, the energy that is consumed in output transmission is 100 pJ/bit/m2 [198].

In addition, the maximum energy consumed in coding and decoding, by using AES encryption, is 0.55 nJ/bit [199].

Attending to Fig. 6.32 Fig. 6.33 and and, a node is involved in the following messages exchanged during an emergency situation in which the node belongs to an affected route:

- It receives a FL Behavior in order to modify the actions that it executes in its usual way of working from the CH.
- It also receives a FL Behavior from the CH for each neighbor that exists in the new route. This is done for malfunction utilities.
- It sends an ACK message to each neighbor. It also receives an ACK from every neighbor.

In order to reduce the energy consumption, the system uses UDP instead of TCP. This decision not only avoids the connection establishment mechanism or congestion control as it is not necessary for this proposal, but also reduces the bits transmitted. With the combination of IEEE 802.15.4 for communicating

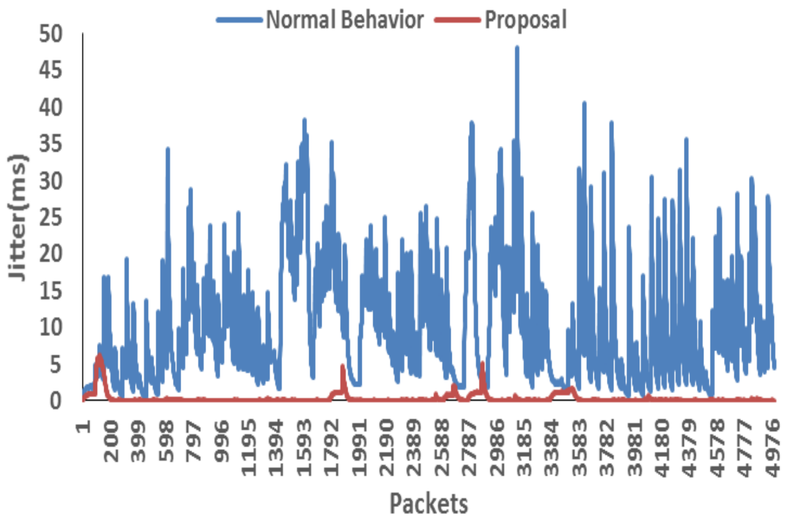


Figure 6.29: Jitter obtained in scenario 4.

the nodes with the CH, IPv6 and UDP the size of the messages is between 80 and 82 bytes.

With this way of communicating we can define the energy consumed in communication as it is expressed in equation (6.8):

$$\begin{aligned} E_{com} = N_{Neig} * S_{Ack} * (E_{bit-tx} + E_{bit-out} + E_{coding}) + \\ + N_{Neig} * S_{Ack} * (E_{bit-rx} + E_{decoding}) + \\ + (N_{Neig} + 1) * S_{Beh} * (E_{bit-rx} + E_{decoding}) \end{aligned} \tag{6.8}$$

Where

N_{Neig} = Number of Neighbors

S_{Ack} = Size of ACK message

S_{Beh} = Size of FL Behavior message

The first addend corresponds to the energy wasted in forwarding the ACK to every neighbor. The second one is related to the received ACKs from the neighbors. Finally, the last addend belongs to the energy consumed in the reception of the behavior messages from the CH.

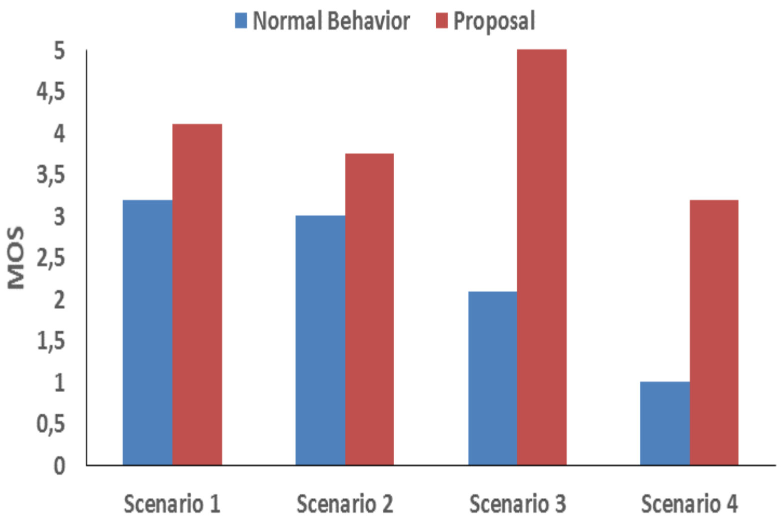


Figure 6.30: QoE obtained in each scenario.

6.4.4 Methodology and Results

In this section, the experiments performed to test our proposal are detailed. They are all executed in a scenario where a smart city is deployed exactly as the one showed in Fig. 6.35. This smart city is treated as an SDN. The equivalent SDN is showed in Fig. 6.36. In this figure, can be seen as the IoT networks are replaced by OpenFlow switches and the different neighbors are converted into PC hosts. The links between the Switches or PCs are the different roads and highways in the city. Fig. 6.36 shows also the different labels and IP addresses given to every node in the SDN. Nodes s1-s6 belong to the city itself and have some hosts connected to them. Switches s3 and s4 represent the inner-city. Furthermore, s7-s14 represents the beltway of the city. The color of the links describes the different scenarios where those roads are involved in the emergency. This is further explained in the next subsection.

The experiments are run by using the emulator Mininet, which provides an easy way to test the SDN by emulating the Switches as Linux Hosts using OpenvSwitch. The urban traffic (emergency and regular traffic) corresponds to packets in the SDN.

The experiment is divided into two parts. The first part compares the behavior of real urban traffic during an emergency in both situations, with the

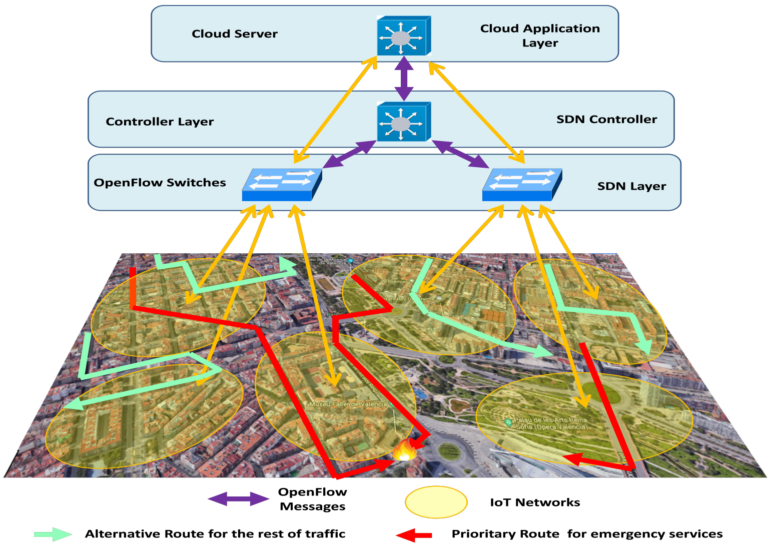


Figure 6.31: Scenario and proposed architecture.

performance of our proposal and without it. The second part measures the energy consumed in three different scenarios depending on the number of nodes and the neighbors in the IoT networks. Each experiment is described in the following subsections.

Traffic Comparison

The traffic in the city is simulated as packets that go through the network. In this first test several multimedia streams are launched in order to emulate normal traffic in a city. The delay is measured as the time the cars need to get to their destination. Then, an emergency occurs, and some emergency traffic is sent to the emergency area. The delay of the emergency resources is also measured.

This experiment is run in two different scenarios. First, the emergency happens, and no algorithm is functioning, so our proposal does not provide any solution. This is the emulation of the usual way of working with emergencies.

On the other hand, in the second scenario, we also measured the delay that both the emergency and the normal traffic have when the proposal provides solutions for emergency priority routes and normal traffic diversion. In order to

Algorithm 6.2 Alert Treatment

```
Given: Alert_ID, Alert_Area, Resources

Alert.Type = Get_Type(Alert_ID)
If Alert.Type == HIGH_RISK
    Local_Resources = Ask for Resources to the Local Security service
    Resources = Resources U Local_Resources
End If
(CH_Set, Flows, Behaviors) = Calculate_Routes(Resources, Alert_Area)
Send the OpenFlow Modification Messages(CH_Set, Flows)
Send Emergency_Notification(CH)
Foreach each IoT Node n in CH_Set
    Send FL Behavior(Behaviors)
End Foreach
```

achieve this, an emergency in the area emulated as the switch s3 is launched. This emergency is emulated as a congestion in the links in red as seen in Fig. 6.36. The links in other colors are not congested. The meaning of these colors is explained in the following experiment and they are not relevant in the first one.

When the emergency is launched, the emergency traffic is sent. In the first scenario, it shares the congested links with the normal traffic along the entire path to s3. However, when the algorithm is running, it forwards the normal traffic through s2.

The dynamic modification of the path used to forward the normal traffic that the algorithm uses, together with the priority of the emergency results, implies a variation of the delay of both types of traffic. This is showed in Fig. 6.37 and Fig. 6.38, where the delay of normal traffic is displayed, and in Fig. 6.39 and Fig. 6.40, where the delay of the emergency traffic is presented. In the normal traffic graphs, the x-axis represents the packet number in the multimedia stream. However, in the emergency traffic, the resources are treated individually, and the x-axis represents the moment in time after the emergency occurs, when the resource is sent. In Fig. 6.37 and Fig. 6.39 the traffic is sent in the first scenario, without algorithm and, in Fig. 6.38 and Fig. 6.40, the algorithm is working.

In Fig. 6.38, the normal traffic is being sent in the first scenario in normal conditions until the packet number 1366 approximately, where the emergency

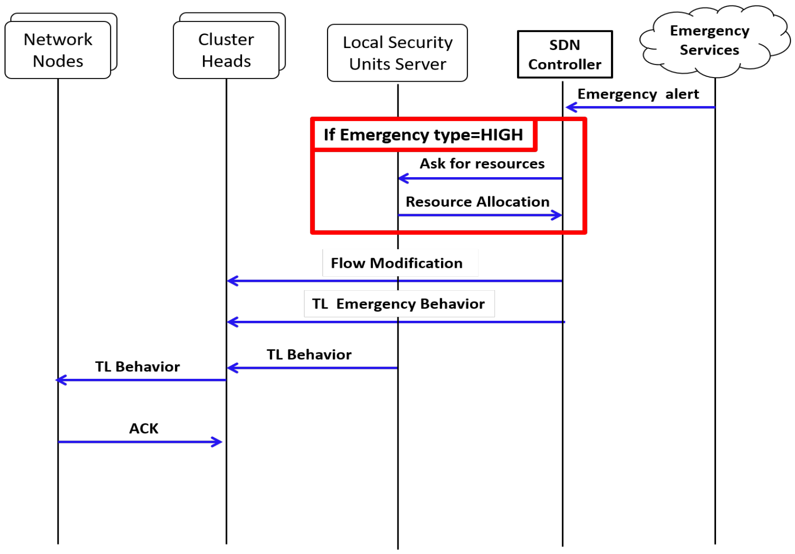


Figure 6.32: Message exchange when alarm is detected.

happens. In Fig. 6.37 the normal traffic is suffering the congestion and then, the algorithm modifies the route paths. Both graphs show the change that delay values experience.

In the first scenario, regarding the normal traffic, it starts with normal values of delay of 20ms and sometimes around 100 (due to the variation of the multimedia bitrate). The average delay, however, is around 10ms. When the emergency occurs, the links are slightly congested and shared with emergency traffic. The average delay increments to more than 130ms.

In the second scenario, the normal traffic experiments a delay just like the delay in scenario 1, but when the algorithm changes the path towards one slower and longer but not congested route, the delay decreases, achieving an average delay of 67ms. The average delay is greater than the delay obtained when there is no emergency in the path, but it is significant lower than the one obtained when there is no algorithm, a 50% of average delay reduction approximately.

Regarding the emergency traffic, Fig. 6.39, related to the first scenario, shows that the average delay is 26ms. When the algorithm is running, in Fig. 6.40, the average delay decreases to 17ms. There is a 33% of average delay reduction.

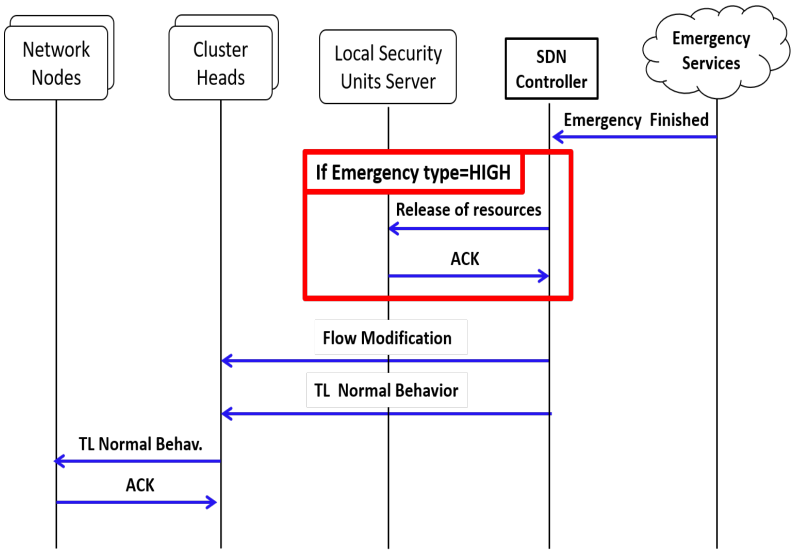


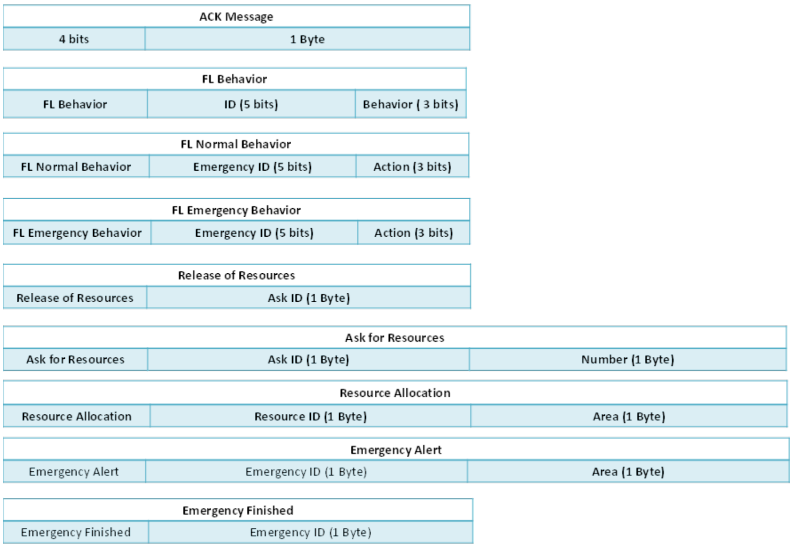
Figure 6.33: Message exchange when alarm is finished.

Energy Consumption

The second experiment, whose aim is to measure the energy consumed in the communication of the nodes, is executed in three different scenarios. These scenarios differ in the size of the emergency in terms of links congested in the network. The links that are affected in each emergency are colored differently as seen in Fig. 6.36. The red ones are affected in the three scenarios, even the third one which is the one used in the previous subsection, where only the red links are affected. Scenario 2 includes the green link. It increments the emergency damage isolating the neighbor connected to s3. Finally, by including the yellow links we obtain scenario 1, where there is a high-risk emergency, and the inner-city paths are closed. The traffic is forwarded through the beltway.

These three scenarios involve a different number of communications between the nodes. The energy consumed per node and the total energy consumed in the nodes due to the emergency has been measured in terms of mJ attending to the equations presented in the previous subsection. Moreover, the number of neighbors has been increased in each individual simulation and the total has been calculated depending on the size of the IoT networks.

In Fig. 6.41 the individual energy consumption per node is showed. The difference between the two last scenarios regarding the individual energy consumed



ACK Message	
4 bits	1 Byte

FL Behavior		
FL Behavior	ID (5 bits)	Behavior (3 bits)

FL Normal Behavior		
FL Normal Behavior	Emergency ID (5 bits)	Action (3 bits)

FL Emergency Behavior		
FL Emergency Behavior	Emergency ID (5 bits)	Action (3 bits)

Release of Resources	
Release of Resources	Ask ID (1 Byte)

Ask for Resources		
Ask for Resources	Ask ID (1 Byte)	Number (1 Byte)

Resource Allocation		
Resource Allocation	Resource ID (1 Byte)	Area (1 Byte)

Emergency Alert		
Emergency Alert	Emergency ID (1 Byte)	Area (1 Byte)

Emergency Finished	
Emergency Finished	Emergency ID (1 Byte)

Figure 6.34: Message Structure.

is only significant when the number of neighbors increases to 5. The bigger difference appears in scenario 1, with much more nodes involved. The increment of energy consumption is linear with the number of neighbors, which implies that the proposal is scalable. The energy in scenario 3 increments from 37mJ with 1 single neighbor to 185mj with 6 Neighbors.

From Fig. 6.42 to Fig. 6.44 are related to total energy consumed. The energy in every figure is calculated with a different number of neighbors. Fig. 6.42 describes the results with 2 neighbors. The x-axis in these graphs is the number of nodes in every IoT network. The increment of the number on nodes increases linearly the total energy consumed in the exchange of messages. The three scenarios described are painted in different colors. The difference between scenario 2 and scenario 3 is not as significant as the difference between scenario 2 and scenario 1. The energy consumed in scenario 1 (6698mJ) with 3 nodes is more than 3 times the energy wasted in scenario 2 (2009mJ). When 10 nodes in the IoT network the difference increases, being 22327mJ the energy in scenario 1 and 6326mJ in scenario 2.

As Fig. 6.43 shows, with 4 neighbors in each IoT network the energy consumed increases, but the results are similar to the presented in Fig. 6.42. Scenario 3 consumes 7566mJ with 10 nodes and scenario 2 12487mJ. Finally, scenario 1 wastes 41622mJ, almost 4 times the energy needed in scenario 2. Moreover,

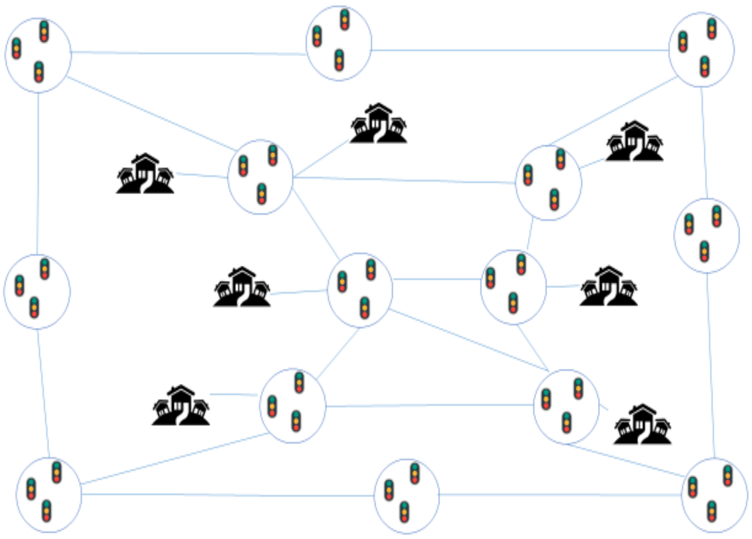


Figure 6.35: Scheme of the city used for testing.

the energy consumed has increased near a 100% from the consumed with 2 neighbors. From example, in scenario 2, with 2 neighbors the energy wasted is between 3349mJ with five nodes and 6698mJ with 10 nodes in every IoT network. However, with 4 neighbors the energy wasted is between 6243mJ and 12487mJ.

Finally, Fig. 6.44 contains the results obtained with 6 neighbors. The results show how the difference between scenario 1 and the rest is greater than before, passing the 60000mJ in scenario 1 and scenarios 2 and 3 consuming between 10000mJ and 20000mJ.

6.4.5 Conclusion

Smart cities provide new applications based on IoT technology. Moreover, SDNs offer the possibility to control the network based on application requirements. In this section, we combine this two technologies in order to take advantage of this possibility and propose a new application to manage emergency resources, based on the routing module introduced in chapter 4. This application uses the adaptability of the SDN in order to forward urban traffic using intelligent traffic lights and other traffic elements that are connected to IoT networks. The SDN controller communicates with cloud and local services

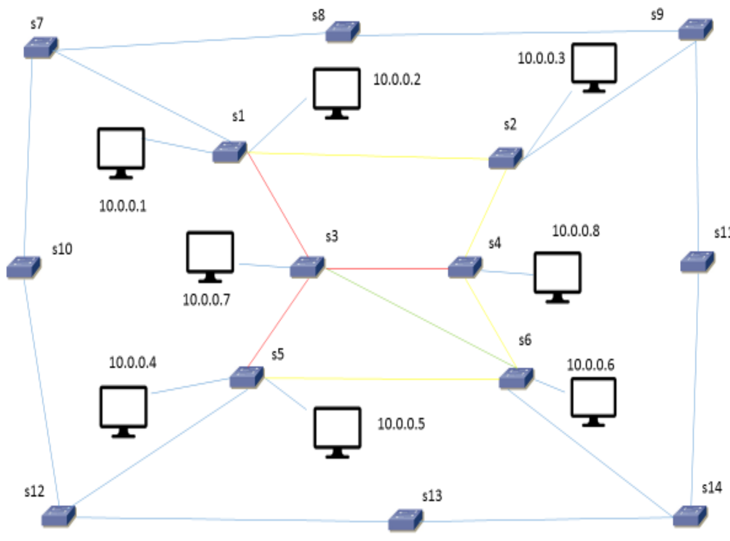


Figure 6.36: SDN corresponding to the Smart City.

so as to gather the data needed to make the best changes in order to manage the emergency.

Experiments show that the average time that emergency resources lose in getting to the emergency area is reduced in 33%, from 26ms to 17ms. This is due to the priority applied and path sharing avoidance. In terms of normal traffic, there is an increment of time when this traffic is forwarded toward a longer alternative path, but this time is lower than the time it takes to get to the destination when the emergency area is becoming congested. The average time difference is 50% between the alternative path and the congested path.

Furthermore, the communication protocol in case of emergency has been tested in terms of energy consumption. Based on the energy model described, experiments illustrate that the energy consumed increases linearly with the number of nodes and neighbors in the IoT networks, which indicates that the proposal is scalable and could be applied in every kind of smart city.

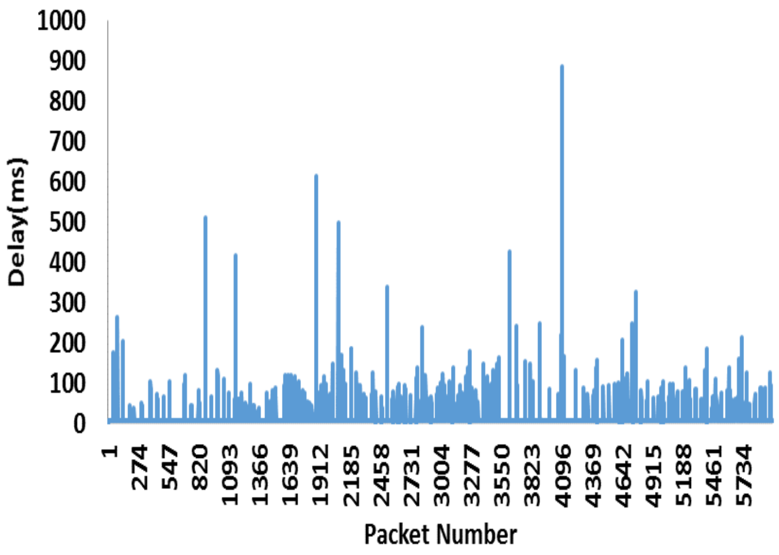


Figure 6.37: Delay in normal traffic sending without system application.

6.5 Conclusion

In this chapter, the different applications of the SDN proposal discussed in this dissertation have been detailed.

For each proposal, the architecture, messages and necessary details have been studied. Moreover, for two of them, a performance evaluation has been included.

These applications have the goal of demonstrating the capability of the proposal to be included in heterogeneous networks. The adaptability of the technologies chosen in the previous chapters helps to use the proposal in different scenarios. In the chapter, WSN, IoT and Smart Cities have been the scenarios where the system has been deployed. For each proposal, some extensions of the protocol have been included to implement new services.

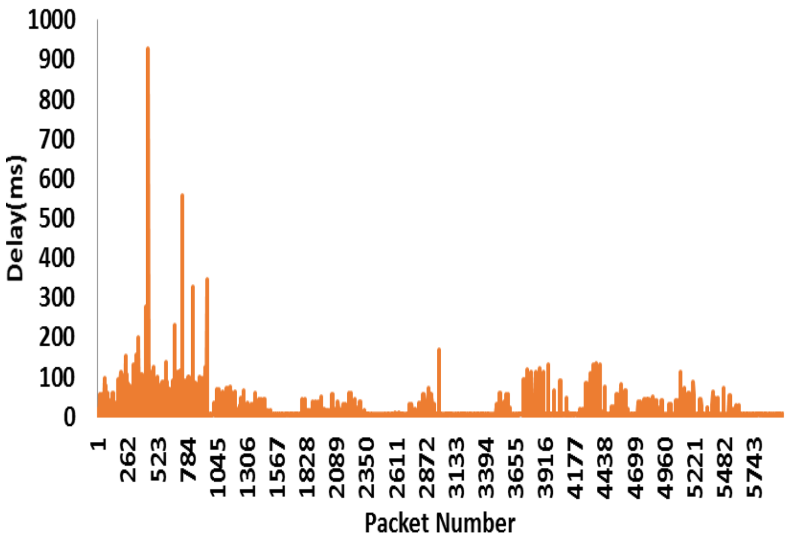


Figure 6.38: Delay in normal traffic sending with system application.

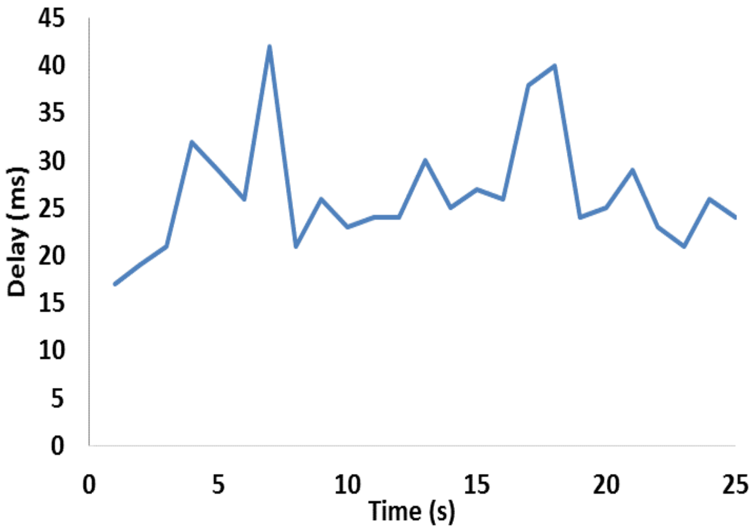


Figure 6.39: Delay in emergency traffic without system application.

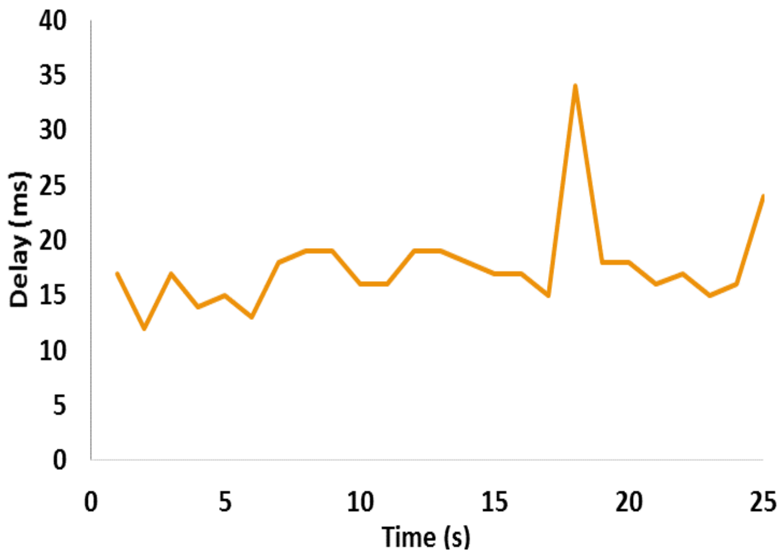


Figure 6.40: Delay in emergency traffic with system application.

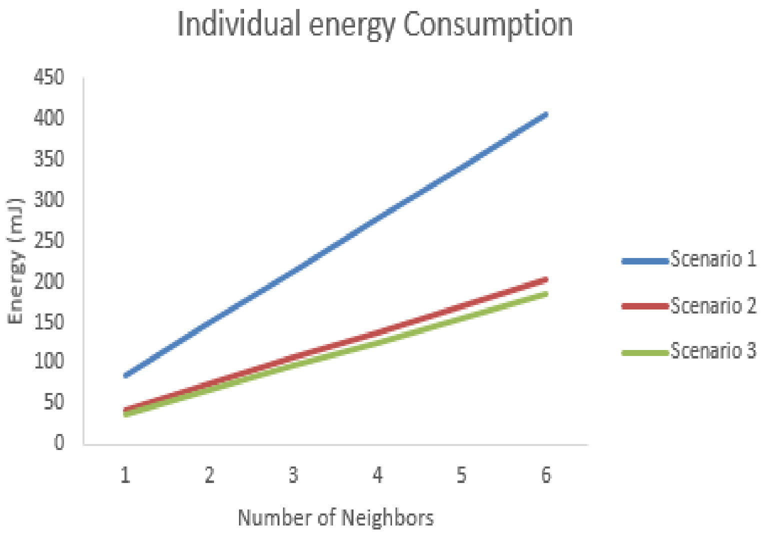


Figure 6.41: Energy consumed per node.

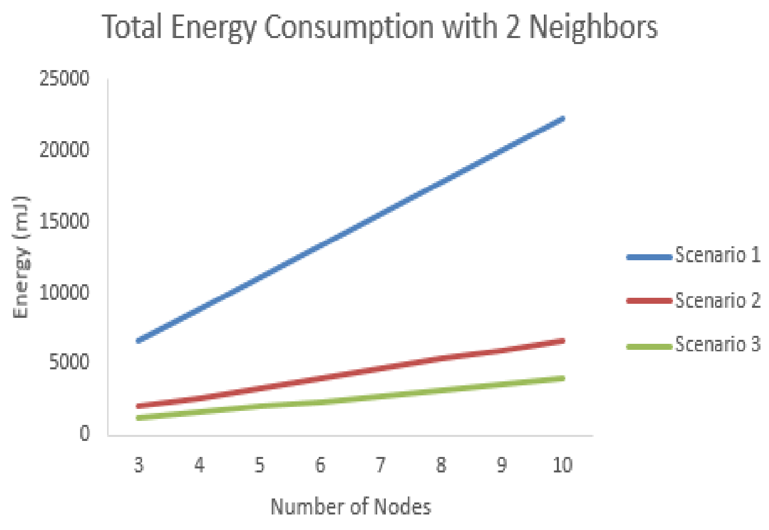


Figure 6.42: Total energy consumed per scenario with 2 neighbors.

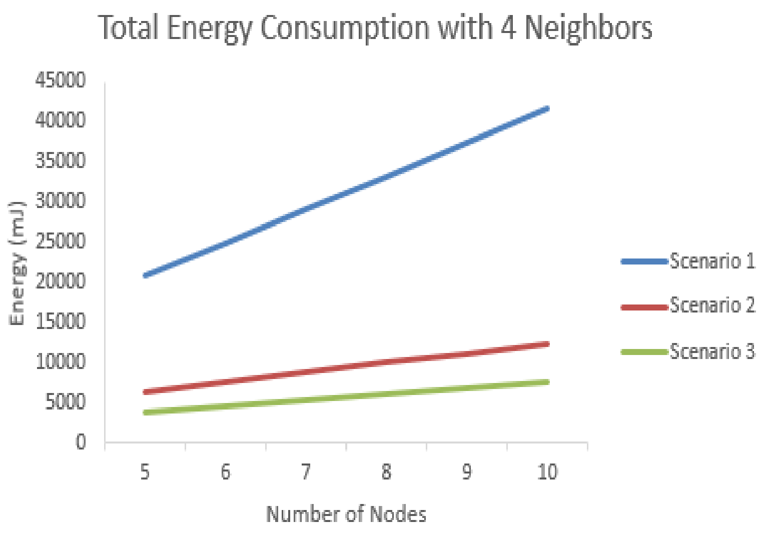


Figure 6.43: Total energy consumed per scenario with 4 neighbors.

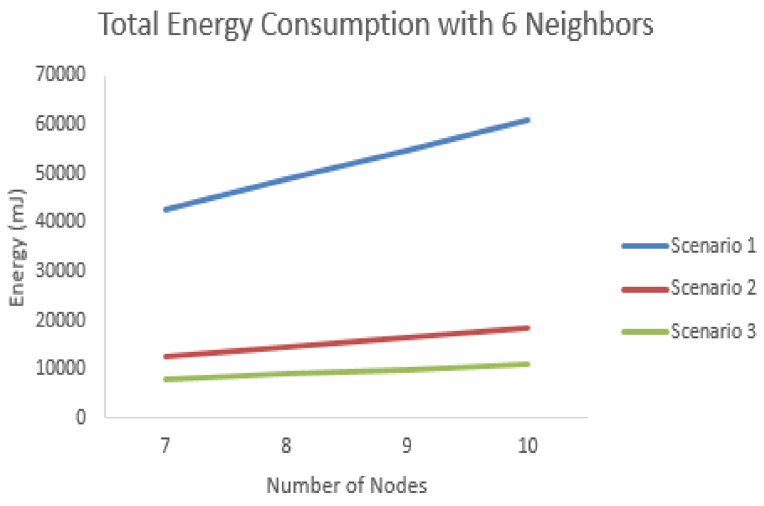


Figure 6.44: Total energy consumed per scenario with 6 neighbors.

Chapter 7

Conclusions

7.1 Introduction

In this chapter, the conclusions obtained from the work presented are discussed. The future research lines that can be followed from this work are also described. Finally, the publications related to the work presented in this dissertation are shown.

7.2 Conclusion

Based on the objectives proposed in chapter 1, it can be concluded that the main goals of this work have been reached.

The introduction of SDN could create the opportunity to bring in new mechanisms and protocols in the network control plane. Thanks to the programmability of this new architecture, new solutions to management problems can be proposed. One of them, the scope of this work, is intelligent routing focused on multimedia transmissions.

Therefore, in this work, the following objectives have been achieved. Firstly, a new software module, a custom SDN controller, has been developed. This custom SDN controller made easier to develop the other software modules and protocols presented in the work. The design of the SDN controller, along with some implementation fragments, was presented in chapter 3. Furthermore, a comparison of resources consumption was performed. In this comparison, the differences in terms of CPU and RAM consumption between SDN controller Floodlight and the custom SDN controller were discussed.

The next objective was to adapt a traditional routing algorithm to SDN. The chosen routing protocol was OSPF. An adaptation of the metric based on QoS measurements was proposed. While most of the traditional OSPF routing protocol was kept, the metric calculation and evaluation were adapted to SDN and multimedia transmissions. This adaptation was tested in the Mininet emulator. The results shown that an increment of the performance, in terms of QoS and QoE, was achieved with the metric change.

Taking into account the final goal of the work, proposing a routing protocol that uses artificial intelligent methods to improve the QoS of multimedia transmissions, the next objective was to study the introduction of AI in routing selection. However, focused on the multimedia problem, some studies about the QoS degradation and which factors could affect to that QoS/QoE measurement were needed.

In chapter 5 the studies show how the QoE could be estimated based on the metrics of the multimedia flows. The most adequate system for QoE estimation was CSVM and RNN. Moreover, the different methods were also used to determine when the traffic is critical and which actions should be performed in the network to avoid a QoE degradation. Again, CNN and CSVM-based methods presented similar results. Nonetheless, the RNN presented a better adaptation to the classification problem. Moreover, the RNN performance based on Root Mean Square Error (RMSE) was the highest one.

Once the traffic classification model was studied, another classification problem was addressed. Based on the MOS QoE metric, the objective was to extract which network and, specially, video parameters affected the QoE the most. The preliminary study shows that there was correlation between some video parameters and the QoE.

After that, a clustering method was carried out to extract the subpatterns for QoE regulation model. Once the subpatterns are extracted, the system is able to obtain video parameters from the network conditions. This allows the system to adapt the video to get the desire QoE based on the current status of the network. Therefore, a robust management system was described next. This system classifies the traffic using the classification method to detect multimedia critical traffic. Then, the subpatters are extracted so as to select which video parameters may improve the QoE of the multimedia transmission if they are modified. This system allows the routing protocol proposed to add an action based on video characteristics modification. In addition, according to the study performed, this action can be important to improve the quality of the transmission due to its high correlation with QoE in case there are no other network-topology-based alternatives.

Once the AI methods were designed and proposed, the routing algorithm was discussed. The AI method to route selection was RL, due to its independence to a training procedure, making easier its deployment in any kind of heterogeneous network. The adaptation of RL method to an SDN routing protocol focused on multimedia transmissions was discussed. The set of actions that the SDN controller can perform when a multimedia degradation is detected were listed and the reward calculation process was detailed, together with the algorithms.

The proposed system was also compared to other alternatives and, then, evaluated. The results shown that the capability of evaluating the network status thanks to the SDN architecture and its central view from the controller, along with the introduction of complex rules based on AI and ML methods, increases the QoE and QoS of the multimedia transmissions. Finally, a reward update system was mentioned. This system allows the routing protocol to introduce a period of time where the impact of the action perform is evaluated based on the change, increase or decrease, of the QoS parameters. The proposed method is a statistic matrix that creates a relation between network actions and QoS parameters that are affected by those actions.

Once the proposal was presented, the next objective to cover was to evaluate the proposal, comparing with traditional protocols. This evaluation was performed and presented at the end of chapter 5. The evaluation was performed in a network using real equipment. The equipment was Openflow-enabled, so both traditional and SDN-based routing protocols could be used.

The evaluation of the proposals, the modification of OSPF, the routing proposal based on RL and the traditional OSPF routing protocol, was performed in different scenarios. These scenarios were designed to test the capabilities of the alternatives when facing a wide set of actions in an SDN core network with heterogeneous end networks and hosts. Depending on the different capabilities and the actions taken by the proposal, the performance and the quality of the multimedia transmission varied. The QoS and the QoE of a set of users, classifying the quality perceived from the transmitted video was compared. The proposal achieved a significant increment of the QoE.

Finally, in chapter 6, a variety of applications of the proposal was presented. The objective of this chapter is to show the huge diversity of applications of the proposal and the different kind of networks where it can be applied. Some of the works where the proposal was introduced in different environments are introduced. From the application to WSN for smart irrigation to the management of emergencies in a smart city or the optimal use of video IoT surveillance systems. In this chapter, the advantage of the use of SDN, thanks to its adaptability to heterogeneous networks, introduces new solutions to current problems. Therefore, the heterogeneous capability of the proposal is demonstrated. In conclusion, this work has accomplished all the previous defined objectives. The proposal achieves an increment of the performance of the multimedia transmission, thanks to the architecture used and the introduction of AI and ML methods. Nevertheless, due to the limitations of the work and the huge scope this proposal can reach, some future works and improvements can be researched. In the next subsection this future line of research is discussed and detailed.

7.3 Problems Faced and Personal Contributions

During the development of this work, several problems appeared. The biggest one may be the lack of an SDN controller that allows easy development of core applications. This is what it caused the development of the custom SDN controller. However, that development brought several problems. During the development, the Mininet emulator caused connectivity problems with the SDN controller. The virtual switches started disconnecting after the first messages. The solution implied to change the socket designs to allow synchronized and parallel communication to each node.

The topology discovery protocol was a solution to the problems raised during the connectivity process. When a node received a flow modification message that could not be accepted, it might reset the connection. That used to cause inconsistencies between the SDN controller and the nodes' states.

Finally, as regard routing, the forwarding of the data until the end host was a problem that had to be addressed. The adding of ARP communications and specific flow entries for the end network had to be added to deliver the data to the specific set of end users.

Regarding the personal contributions of this work, it has caused a feeling of skill development in me. Working in a new technology with really few previous works forced me to have a strong control about it and about the SDN controller I personally developed. Therefore, my ability for analyzing problems and designing protocols and software tools were tested and, eventually, improved. Finally, all the difficulties made me not only stronger but also more constant in the work task execution. An improvement that I will definitely need in the future.

7.4 Future Research

The work presented in this dissertation has several lines where continue working and researching. In this section, some of them are discussed.

As regards the SDN controller, updating the system to add compatibility with newer versions of Openflow is one of the main aspects of improvement. However, in terms of research, there are others. For instance, some evaluations of the performance of the SDN controller depending on the software design underlying may be an interesting research. Since the application of the software is the management of network nodes, scalability and throughput are the main metrics. Therefore, applying different designs can help us to determine the best way in terms of performance.

The AI methods proposed in this work can also be improved. Further studies and comparison between different techniques can be performed. Since the design of the architecture is modular, the replacement of the AI method does not imply a big change in the routing protocol.

Furthermore, the reward update system is only slightly addressed in this work. The development and evaluation of applying a heuristic method, or a ML method to solve this problem is a future research line that will be addressed once this work is published.

Virtualization techniques can be also added to the proposal. The study of applying NFV to the routing proposal may be a future line of research. This virtualization could provide virtual SDN organized by ML protocols, or virtual machines, which virtualize SDN, interconnected and managed by AI or ML protocols.

The routing protocol proposed in this work could be adapted to other technologies such as mobile networks or MANETS. The use of AI techniques to determine interconnection could bring improvements to the spectrum use or the energy consumed by the transmission.

Finally, another research line, where a lot of work is needed, is the integration and enhancement of the proposal to manage IoT networks. From the architecture and the base of this project, an intelligent management system can be designed to interconnect IoT and WSN. Therefore, protocols to allow the interconnection of the different technologies could be developed. These protocols can use the AI methods proposed, or some news, to provide the maximum QoE from the IoT networks. Moreover, the definition of a metric similar to QoE to interconnected IoT networks should be defined to allow the intelligent methods to estimate the outcome of the actions performed by the protocol.

Currently, we are working on applying SDN and AI to IoT in order to analyze and determine critical IoT traffic. This is a continuation of the work done in this dissertation. Once the critical IoT traffic is classified, the routing protocol can be enhanced by adding concrete actions and adapting it to other kind of traffic and services.

7.5 Publications derived from the PhD Thesis

The following publications are derived from the research work presented in this dissertation. The papers that are directly related with the dissertation are the following ones:

A. Rego, A. Canovas, J. M. Jiménez and J. Lloret, "An Intelligent System for Video Surveillance in IoT Environments," in IEEE Access, vol. 6, pp. 31580-31598, 2018, doi: 10.1109/ACCESS.2018.2842034. **Impact Factor: 4.098, Q1.**

Albert Rego, Laura Garcia, Sandra Sendra, Jaime Lloret, “Software Defined Network-based control system for an efficient traffic management for emergency situations in smart cities”, Future Generation Computer Systems, vol. 88, pp. 243-253, 2018. https://doi.org/10.1016/j.future.2018.05.054. **Impact Factor: 6.125, Q1.**

Rego, A., Sendra, S., Jimenez, J.M. et al. Dynamic metric OSPF-based routing protocol for Software Defined Networks. Cluster Comput 22, 705–720 (2019). https://doi.org/10.1007/s10586-018-2875-7 **Impact Factor: 3.458, Q1.**

Rego, A, Sendra, S, Garcia, L, Lloret, J. Adapting reinforcement learning for multimedia transmission on SDN. Trans Emerging Tel Tech. 2019; 30:e3643. https://doi.org/10.1002/ett.3643 **Impact Factor: 1.5945, Q3.**

S. Egea, **A. Rego** Mañez, B. Carro, A. Sánchez-Esguevillas and J. Lloret, "Intelligent IoT Traffic Classification Using Novel Search Strategy for Fast-Based-Correlation Feature Selection in Industrial Environments," in IEEE Internet of Things Journal, vol. 5, no. 3, pp. 1616-1624, June 2018, doi: 10.1109/JIOT.2017.2787959.t **Impact Factor: 9.936, Q1.**

Alejandro Canovas, **Albert Rego**, Oscar Romero, Jaime Lloret, "A robust multimedia traffic SDN-Based management system using patterns and models of QoE estimation with BRNN", Journal of Network and Computer Applications, vol. 150, ISSN 1084-8045, 2020. https://doi.org/10.1016/j.jnca.2019.102498. **Impact Factor: 5.57, Q1.**

Jimenez, JM.; Romero Martínez, JO.; **Rego Mañez, A.**; Lloret, J. "Analyzing the Performance of Software Defined Networks vs Real Networks", International Journal On Advances in Networks and Services. Vol. 9(3-4), pp. 107-116, 2016. http://hdl.handle.net/10251/83652.

Jimenez, JM.; Romero Martínez, JO.; **Rego, A.**; Dilendra, A.; Lloret, J., "Study of Multimedia Delivery over Software Defined Networks", Network Protocols and Algorithms. Vol. 7(4), pp. 37-62, 2016. doi:10.5296/npa.v7i4.8794.

These are the publications in conferences:

A. Rego, S. Sendra, J. M. Jimenez and J. Lloret, "OSPF routing protocol performance in Software Defined Networks," 2017 Fourth International Conference on Software Defined Systems (SDS), Valencia, 2017, pp. 131-136, doi: 10.1109/SDS.2017.7939153.

A. Rego, L. Garcia, S. Sendra and J. Lloret, "Software defined networks for traffic management in emergency situations," 2018 Fifth International Conference on Software Defined Systems (SDS), Barcelona, 2018, pp. 45-51, doi: 10.1109/SDS.2018.8370421.

A. Rego-Mañez, S. Sendra, J. L. García-Navas and J. Lloret, "Managing a Multi-device Multimedia Service Using Software Defined Networks," 2019 Sixth International Conference on Software Defined Systems (SDS), Rome, Italy, 2019, pp. 221-228, doi: 10.1109/SDS.2019.8768560.

S. Sendra, **A. Rego**, J. Lloret, J. M. Jimenez and O. Romero, "Including artificial intelligence in a routing protocol using Software Defined Networks," 2017 IEEE International Conference on Communications Workshops (ICC Workshops), Paris, 2017, pp. 670-674, doi: 10.1109/ICCW.2017.7962735.

D. Sarabia-Jácome, **A. Rego**, S. Sendra and J. Lloret, "Energy consumption in software defined networks to provide service for mobile users," 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, 2017, pp. 1453-1458, doi: 10.1109/IWCMC.2017.7986498.

Jose M Jimenez, Oscar Romero, **Albert Rego**, Avinash Dilendra, Jaime Lloret, "Performance study of a software defined network emulator", The Eleventh International Conference on Internet Monitoring and Protection,(ICIMP 2016), 2016.

References

[1] Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper. Doc. ID: 1551296909190103.Updated: February 27, 2019. In Cisco website. Available at: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html [Last access: March 4, 2020]

[2] Internet usage statistics (June 30, 2018). In Internet world stats. Available at: https://www.internetworldstats.com/stats.htm [Last access: March 4, 2020]

[3] A. Schomer, "HBO is making moves on local market content", May 14, 2018. (On line article). Available at: https://www.businessinsider.com/hbo-vs-netflix-subscribers-and-revenue-2016-2?IR=T [Last access: March 4, 2020]

[4] SHENKER, Scott; PARTRIDGE, Craig; GUERIN, Roch, "Specification of guaranteed quality of service", RFC 2212, September, 1997.

[5] ALRESHOODI, Mohammed; WOODS, John, "Survey on QoE QoS correlation models for multimedia services". arXiv preprint arXiv:1306.0221, 2013.

[6] "Software-Defined Networking (SDN) Definition", Available at: https://www.opennetworking.org/sdn-definition/ [Last access: March 4, 2020]

[7] XIA, Wenfeng, et al, "A survey on software-defined networking", IEEE Communications Surveys & Tutorials, 2014, vol. 17, no. 1, pp. 27-51.

[8] HALEPLIDIS, Evangelos, et al, “Software-defined networking (SDN): Layers and architecture terminology”, RFC 7426. IRTF, 2015.

[9] MCKEOWN, Nick, et al, “OpenFlow: enabling innovation in campus networks”, ACM SIGCOMM Computer Communication Review, vol. 38, no 2, pp. 69-74. 2008.

[10] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, Scott Shenker, “NOX: towards an operating system for networks”, ACM SIGCOMM Computer Communication Review, 2008, vol. 38, no. 3, pp. 105-110. https://doi.org/10.1145/1384609.1384625

[11] VMware and Nicira, Available at:
https://www.vmware.com/company/acquisitions/nicira.html
[Last access: March 4, 2020]

[12] “NOX Classic”, Available at: https://github.com/noxrepo/nox-classic
[Last access: March 4, 2020]

[13] “NOX”, Available at: https://github.com/noxrepo/nox/
[Last access: March 4, 2020]

[14] “POX Controller”, Available at: https://noxrepo.github.io/pox-doc/html/
[Last access: March 4, 2020]

[15] “Beacon”, Available at:
https://openflow.stanford.edu/display/Beacon/Home classic
[Last access: March 4, 2020]

[16] Erickson, David, “The Beacon OpenFlow Controller”, HotSDN 2013 - Proceedings of the 2013 ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, pp. 13-18, Hong Kong, China, 2013.
doi: 10.1145/2491185.2491189.t

[17] “MAESTRO-Platform”, Available at: http://code.google.com/p/maestro-platform/ [Last access: March 4, 2020]

[18] “FloodLight”, Available at: http://www.projectfloodlight.org/floodlight/
[Last access: March 4, 2020]

[19] “Big Switch Networks”, Available at: https://www.bigswitch.com/
[Last access: March 4, 2020]

[20] “How to Write a Module in Floodlight”, Available at: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343513/How+to+Write+a+Module [Last access: March 4, 2020]

[21] “The Linux foundation”, Available at: https://www.linuxfoundation.org/ [Last access: March 4, 2020]

[22] “OpenDaylight”, Available at: http://www.opendaylight.org/ [Last access: March 4, 2020]

[23] J. Medved, R. Varga, A. Tkacik and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, Sydney, NSW, 2014, pp. 1-6. doi: 10.1109/WoWMoM.2014.6918985

[24] “YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)”, Available at: https://tools.ietf.org/html/rfc6020 [Last access: March 4, 2020]

[25] “ONAP”, Available at: https://www.onap.org/ [Last access: March 4, 2020]

[26] “OpenStack”, Available at: https://www.openstack.org/ [Last access: March 4, 2020]

[27] “OPNFV”, Available at: https://www.opnfv.org/ [Last access: March 4, 2020]

[28] “Ryu”, Available at: http://osrg.github.io/ryu/ [Last access: March 4, 2020]

[29] S. Asadollahi, B. Goswami and M. Sameer, "Ryu controller’s scalability experiment on software defined networks," 2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), Bangalore, 2018, pp. 1-5. doi: 10.1109/ICCTAC.2018.8370397

[30] “Trema”, Available at: http://trema.github.io/trema/ [Last access: March 4, 2020]

[31] “Cisco Open SDN Controller”, Available at: https://www.cisco.com/c/en/us/products/ cloud-systems-management/open-sdn-controller/index.html [Last access: March 4, 2020]

[32] “Open SDN Controller Datasheet”, Available at:
https://www.cisco.com/c/en/us/products/collateral/
cloud-systems-management/open-sdn-controller/datasheet-c78-733458.html
[Last access: March 4, 2020]

[33] “Aruba VAN SDN Controller Data Sheet”, Available at:
https://tdhpe.techdata.eu/Documents/Aruba/
DS_VAN_SDN.pdf?epslanguage=en [Last access: March 4, 2020]

[34] “RouteFlow”, Available at: https://cpqd.github.io/RouteFlow/ [Last access: March 4, 2020]

[35] “RouteFlow documentation”, Available at:
https://sites.google.com/site/routeflow/home [Last access: March 4, 2020]

[36] “FlowVisor”, Available at:https://groups.geni.net/geni/wiki/FlowVisor
[Last access: March 4, 2020]

[37] S. T. Sany, S. Shashidhar, M. P. Gilesh and S. D. M. Kumar, "Performance evaluation and assessment of FlowVisor," 2016 International Conference on Information Science (ICIS), Kochi, pp. 222-227, 2016. doi: 10.1109/INFOSCI.2016.7845331

[38] Nayak, Ankur & Reimers, Alex & Feamster, Nick & Clark, Russell, “Resonance: Dynamic access control for enterprise networks”, Proceedings of the 1st ACM Workshop on Research on Enterprise Networking, pp. 11-18, Barcelona, Spain, 2009. doi: 10.1145/1592681.1592684.

[39] Rotsos, Charalampos & Sarrar, Nadi & Uhlig, Steve & Sherwood, Rob & Moore, Andrew, “OFLOPS: An open framework for OpenFlow switch evaluation”, Lecture Notes in Computer Science, vol. 7192, 2012. doi: 10.1007/978-3-642-28537-0_9.

[40] WANG, Tao, et al, “Dynamic SDN controller assignment in data center networks: Stable matching with transfers”, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications. IEEE, pp. 1-9, San Francisco, CA, 2016.

[41] “Google Brings SDN to the Public Internet”, Available at:
https://www.sdxcentral.com/articles/news/google-brings-sdn-public-internet
/2017/04/ [Last access: March 5, 2020]

[42] "Vodafone Ready Network", Available at:
https://www.vodafone.com/business/solutions/fixed-connectivity/ready-network [Last access: March 5, 2020]

[43] "Vodafone SDN Research Report 2019", Available at:
https://www.vodafone.com/business/news-and-insights/white-paper/SDN-research-report-2019 [Last access: March 5, 2020]

[44] Corrado Rametta, Gabriele Baldoni, Alfio Lombardo, Sergio Micalizzi, Alessandro Vassallo, "S6: a Smart, Social and SDN-based Surveillance System for Smart-cities", Procedia Computer Science, Vol. 110, pp. 361-368, 2017. https://doi.org/10.1016/j.procs.2017.06.078.

[45] F. Olivier, G. Carlos and N. Florent, "," 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Blumenau, pp. 342-347, Blumenau, Brazil, 2015. doi: 10.1109/IMIS.2015.52

[46] R. Vilalta, A. Mayoral, D. Pubill, R. Casellas, R. Martínez, J. Serra, C. Verikoukis, and R. Muñoz, "End-to-End SDN Orchestration of IoT Services Using an SDN/NFV-enabled Edge Node", 2016 Optical Fiber Communications Conference and Exhibition (OFC), pp. 1-3, Anaheim, CA, 2016.

[47] Trivisonno, R., Guerzoni, R., Vaishnavi, I., and Soldani, D., "SDN-based 5G mobile networks: architecture, functions, procedures and backward compatibility", Trans. Emerging Tel. Tech., Vol. 26, pp. 82– 92.
doi: 10.1002/ett.2915

[48] M. Taha, L. García, J.M. Jimenez and J. Lloret, "SDN-based throughput allocation in wireless networks for heterogeneous adaptive video streaming applications", 13th International Wireless Communications and Mobile Computing Conference, Valencia, Spain, 26-30 June 2017; 963-968. Doi: 10.1109/IWCMC.2017.7986416.

[49] Jose Miguel Jimenez, Oscar Romero, Albert Rego, Avinash Dilendra, Jaime Lloret, "Study of Multimedia Delivery over Software Defined Networks", Network Protocols and Algoritms, vol. 7, pp. 37-62, 2015.

[50] Vandana and H. D. Kallinatha, "An Enhanced method for Streaming Multimedia data to achieve the Quality of Service Support using SDN. 2nd International Conference on Emerging Computation and Information Technologies", Tumakuru, India, 15-16 December, 2017; pp. 1-7.
Doi: 10.1109/ICECIT.2017.8453366.

[51] A. Rego, S. Sendra, J. M. Jimenez and J. Lloret, “OSPF routing protocol performance in Software Defined Networks”, 2017 Fourth International Conference on Software Defined Systems (SDS), Valencia, 2017; pp. 131-136. Doi: 10.1109/SDS.2017.7939153

[52] M. Karakus and A. Durresi, “Quality of Service (QoS) in Software Defined Networking (SDN): A survey”, Journal of Network and Computer Applications, vol. 80, pp.200-218, 2017. Quality of Service (QoS) in Software. Doi: 10.1016/j.jnca.2016.12.019.

[53] D. E. O’Leary, "Artificial Intelligence and Big Data", IEEE Intelligent Systems, vol. 28, no. 2, pp. 96-99, March-April 2013. doi: 10.1109/MIS.2013.39

[54] M. O. Riedl and A. Zook, "AI for game production", 2013 IEEE Conference on Computational Intelligence in Games (CIG), Niagara Falls, ON, 2013, pp. 1-8. doi: 10.1109/CIG.2013.6633663

[55] John Stoitsis, Ioannis Valavanis, Stavroula G. Mougiakakou, Spyretta Golemati, Alexandra Nikita, Konstantina S. Nikita, “Computer aided diagnosis based on medical image processing and artificial intelligence methods”, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 569, no. 2, pp. 591-595, 2006. https://doi.org/10.1016/j.nima.2006.08.134.

[56] Hu, F., Hao, Q., Bao, K.,"A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation", Communications Surveys & Tutorials, IEEE, vol. 16, no. 4, pp. 2181 – 2206, 2014.
http://dx.doi.org/10.1109/COMST.2014.2326417

[57] Scott-Hayward, S., O’Callaghan, G., Sezer, S., "SDN Security: A Survey", IEEE SDN for Future Networks and Services (SDN4FNS 2013), 11-13 Nov. 2013, Trento, Italy, pp. 1-7.
http://dx.doi.org/10.1109/SDN4FNS.2013.6702553

[58] Kreutz, D., Ramos, F.M.V., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., Uhlig, S., "Software-Defined Networking: A Comprehensive Survey", Proceedings of the IEEE, vol. 103, no. 1, Jan. 2015, pp. 14-76.

[59] Kaur, K., Singh, J., Ghumman, N. S., “Mininet as Software Defined Networking Testing Platform”, International Conference on Communication, Computing & Systems (ICCCS. 2014), 20 Feb - 21 Feb 2014, Chennai, India.

[60] Paasch, C., Ferlin, S., Alay, O., Bonaventure, O., “Experimental Evaluation of Multipath TCP Schedulers”, ACM SIGCOMM Capacity Sharing Workshop (CSWS), 2014. ACM., August 18, 2014, Chicago, IL, USA. http://dx.doi.org/10.1145/2630088.2631977

[61] de Oliveira, R.L.S., Shinoda, A.A., Schweitzer, C.M., Rodrigues, P. L., "Using Mininet for Emulation and Prototyping Software-Defined Networks", IEEE Communications and Computing (COLCOM 2014), 4-6 June 2014, Bogota, Colombia, pp. 1-6. http://dx.doi.org/10.1109/ColComCon.2014.6860404

[62] Keti, F., Askar, S., "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments", 6th International Conference on. IEEE Intelligent Systems, Modelling and Simulation (ISMS 2015), Kuala Lumpur, Malaysia, 2015. pp. 205-210. http://dx.doi.org/10.1109/ISMS.2015.46

[63] Azizi, M., Benaini, R., Ben Mamoun M., "Delay Measurement in Openflow-Enabled MPLS-TP Network", Modern Applied Science, vol. 9, no. 3, 2015, Canadian Center of Science and Education, 2015. http://dx.doi.org/10.5539/mas.v9n3p90

[64] Gupta, M., Sommers, J., Barford, P. "Fast, accurate simulation for SDN prototyping", 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, 12-15 August, 2013, Hong Kong. http://dx.doi.org/10.1145/2491185.2491202

[65] Panwaree, P., Kim, J., Aswakul, C., "Packet Delay and Loss Performance of Streaming Video over Emulated and Real OpenFlow Networks.", The 29th International Technical Conference on Circuit/Systems Computers and Communications (ITC-CSCC), Phuket, Thailand, July 1-4, 2014

[66] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, and S. A. Mehdi, “An Architectural Evaluation of SDN Controllers”. In proceedings of the IEEE ICC 2013 - Next-Generation Networking Symposium, Budapest, June 9-13, 2013.

[67] Alexander Shalimov, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky, “Advanced study of SDN/OpenFlow controllers”, Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR ’13). Association for Computing Machinery, New York, NY, USA, Article 1, pp. 1–6, 2013. DOI:https://doi.org/10.1145/2556610.2556621

[68] Bholebawa, I.Z., Dalal, "U.D. Performance Analysis of SDN/OpenFlow Controllers: POX Versus Floodlight", Wireless Pers Commun, 2018, vol. 98, pp. 1679–1699. https://doi.org/10.1007/s11277-017-4939-z

[69] C. Fancy and M. Pushpalatha, "Performance evaluation of SDN controllers POX and floodlight in mininet emulation environment", 2017 International Conference on Intelligent Sustainable Systems (ICISS), Palladam, 2017, pp. 695-699.
doi: 10.1109/ISS1.2017.8389262

[70] Z. K. Khattak, M. Awais and A. Iqbal, "Performance evaluation of OpenDaylight SDN controller", 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), Hsinchu, 2014, pp. 671-676. doi: 10.1109/PADSW.2014.7097868

[71] Rowshanrad, S., Abdi, V., & Keshtgari, M. (2016). PERFORMANCE EVALUATION OF SDN CONTROLLERS: FLOODLIGHT AND OPENDAYLIGHT. IIUM Engineering Journal, vol. 17, no. 2, 47-57.
https://doi.org/10.31436/iiumej.v17i2.615

[72] R. K. Arbettu, R. Khondoker, K. Bayarou and F. Weber, "Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers," 2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks), Montreal, QC, 2016, pp. 37-44.
doi: 10.1109/NETWKS.2016.7751150

[73] D. LeBlanc, "The STRIDE Threat Model.". Available at:
https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx [Last access: March 17, 2020]

[74] A. Cánovas et al., "Multimedia Data Flow Traffic Classification Using Intelligent Models Based on Traffic Patterns", IEEE Network, vol. 32, no. 6, pp. 100-107, 2018.

[75] Z. Kotevski and P. Mitrevski, "Performance Assessment of Metrics for Video Quality Estimation," Proc. Conf. Info., Commun. Energy Systems and Technologies, pp. 693–96, June 23–26, 2010, Ohrid, Macedonia.

[76] H. Zhang et al., "DeepQoE: A Unified Framework for Learning to Predict Video QoE", IEEE International Conference on Multimedia and Expo (ICME), 23-27 July 2018, San Diego, USA.

[77] M. Amiri et al., “Game-Aware and SDN-Assisted Bandwidth Allocation for Data Center Networks”, IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), 10-12 April 2018, Miami, USA.

[78] V. Vasilev et al., “Predicting QoE Factors with Machine Learning”, IEEE International Conference on Communications (ICC), 20-24 May 2018, Kansas, USA.

[79] A. Floris et al., “Managing the Quality of Experience in the Multimedia Internet of Things: A Layered-Based Approach”, IEEE International Conference on Communication Workshop (ICCW), 8–12 June 2015, London, UK.

[80] M. López-Martíne et al., “Deep Learning Model for Multimedia Quality of Experience Prediction Based on Network Flow Packets”, IEEE Communications Magazine, vol. 56, no. 9, pp. 110-117, 2018.

[81] K. Seshadrinathan et al., “Study of Subjective and Objective Quality Assessment of Video,” IEEE Trans. Image Processing, vol. 19, no. 6, pp. 1427–41, 2010.

[82] X. Huang et al., “Deep Reinforcement Learning for Multimedia Traffic Control in Software Defined Networking”, IEEE Network, vol. 32, no. 6, pp. 35-41, 2018.

[83] T. T. T. Nguyen and G. Armitage, “A Survey of Techniques for Internet Traffic Classification Using Machine Learning,” IEEE Communications Surveys & Tutorials, vol. 10, no. 4, pp.56–76, 2008.

[84] M. Soysal and E. G. Schmidt, “Machine Learning Algorithms for Accurate Flow-Based Network Traffic Classification: Evaluation and Comparison,” Performance Evaluation, vol. 67, no. 6, pp. 451–67, 2010.

[85] A. W. Moore and D. Zuev, “Internet Traffic Classification Using Bayesian Analysis Techniques,” Proc. 2005 ACM SIGMETRICS Int’l. Conf. Measurement and Modeling of Computer Systems, Banff, Alberta, Canada, June 6–10, pp. 5–60, 2005.

[86] A. Tongaonkar et al., “Towards Self-Adaptive Network Traffic Classification,” Computer Communications, vol. 56, no. 1, pp. 35–46, 2015.

[87] M.S. Mushtaq, “Empirical study based on machine learning approach to assess the QoS/QoE correlation”, 17th European Conference on Networks and Optical Communications, 2012, Spain.

[88] R. Shalala, “Video QoE Prediction Based on User Profile”, International Conference on Computing, Networking and Communications (ICNC 2018), Maui, USA.

[89] L. Huixian et al., “A Novel Traffic Classification Algorithm using Machine Learning”, IEEE International Conference on Broadband Network & Multimedia Technology, 2009, Beijing, China.

[90] S. Zaner, “Practical Machine Learning Based Multimedia Traffic Classification for Distributed QoS Management”, IEEE 36th Conference on Local Computer Networks, 2011, Bonn, Germany.

[91] M. Taha, L. Garcia, J. M. Jimenez, J. Lloret, "SDN-based throughput allocation in wireless networks for heterogeneous adaptive video streaming applications," 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 26-30 June, 2017, Valencia, Spain, https://doi.org/10.1109/IWCMC.2017.7986416

[92] O. Awobuluyi, J. Nightingale, Q. Wang and J. M. Alcaraz-Calero. Video Quality in 5G Networks: Context-Aware QoE Management in the SDN Control Plane. IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, Liverpool, UK, 26-28 October 2015; 1657-1662.
doi: 10.1109/CIT/IUCC/DASC/PICOM.2015.250.

[93] J. Castillo, A. Neto, F. Silva, P. Fonsi, R. Aguiar, A. Ishimori, F. Farias and A. Abelém. Additions to the ETArch Control Plane to Support Multimedia QoS-Guaranteed Content Transport over OpenFlow-Enabled SDN Future Internet Systems. 6th IEEE International Workshop on Management of Emerging Networks and Services, Austin, TX, USA, 8-12 December 2014; 172-177. Doi: 10.1109/GLOCOMW.2014.7063426.

[94] A. A. Barakabitze, L. Sun, I. Mkwawa and E. Ifeachor. A Novel QoE-Aware SDN-enabled, NFV-based Management Architecture for Future Multimedia Applications on 5G Systems. Proceedings of the 8th International Conference on Quality of Multimedia Experience, Lisbon, Portugal, 6-8 June 2016.

[95] T. Lin, Y. Hsu, S. Kao and P. Chi. OpenE2EQoS: Meter-based Method for End-to-End QoS of Multimedia Services over SDN. IEEE 27th International Symposium on Personal, Indoor and Mobile Radio Communications, Valencia, Spain, 4-8 September 2016; 1-6. Doi: 10.1109/PIMRC.2016.7794972.

[96] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic and P. Dely. Towards QoE-driven Multimedia Service Negotiation and Path Optimization with Software Defined Networking. 20th International Conference on Software, Telecommunications and Computer Networks, Split, Croatia, 11-13 September 2012; 1-5.

[97] H. E. Egilmez, S. T. Sane, K. T. Bagci and M. Tekalp. OpenQoS: An OpenFlow Controller Design for Multimedia Delivery with End-to-End Quality of Service over Software-Defined Networks. Proceedings of the 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference, Hollywood, CA, USA, 3-6 December 2012; 1-8.

[98] E. Grigoriou, A. A. Barakabitze, L. Atzori, L. Sun and V. Pilloni, An SDN-approach for QoE management of multimedia services using resource allocation. IEEE ICC 2017 Communications Software, Services, and Multimedia Applications Symposium, Paris, France, 21-25 May 2017; 1-7. Doi: 10.1109/ICC.2017.7997261.

[99] K. A. Noghani and M. O. Sunay, “Streaming Multicast Video over Software-Defined Networks” Proceedings of the IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems (MASS) (2014), 28-30 Oct. 2014, pages 551-556, 2014, doi: 10.1109/MASS.2014.125

[100] H. Nam, K. Kim, J. Y. Kim and H. Schulzrinney, “Towards QoE-aware Video Streaming using SDN” Global Communications Conference (GLOBECOM), Dec. 2014, pp 1317-1322, 2014, doi: 10.1109/GLOCOM.2014.7036990

[101] H. E. Egilmez, S. T. Dane, K. Tolga Bagci and A. Murat Tekalp, “OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks” Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific, Hollywood (USA), pp 1-8, 3-6 Dec. 2012.

[102] H. E. Egilmez, and A. M. Tekalp, ”Distributed QoS Architectures for Multimedia Streaming Over Software Defined Networks” Multimedia, IEEE Transactions on , vol. 16 , no. 6 , Sept 2014, pp. 1597 – 1609, 2014; doi:10.1109/TMM.2014.2325791

[103] Mehrdad Hosseinnejad Yami, Farhad Pakdaman, and Mahmoud Reza Hashemi. 2020. SARA-SDN: state aware resource allocation in SDN to improve QoE in cloud gaming. In Proceedings of the 25th ACM Workshop on Packet Video (PV ’20). Association for Computing Machinery, New York, NY, USA, pp. 8–14, 2020.DOI:https://doi.org/10.1145/3386292.3397118

[104] E. Liotou, G. Tseliou, K. Samdanis, D. Tsolkas,, F. Adelantado, and C. Verikoukis, “An SDN QoE-Service for dynamically enhancing the performance of OTT applications”. In 2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX), pp. 1-2, Pylos-Nestoras, 2015.

[105] A. Martin et al., "Network Resource Allocation System for QoE-Aware Delivery of Media Services in 5G Networks," in IEEE Transactions on Broadcasting, vol. 64, no. 2, pp. 561-574, 2018.

[106] Mu, Mu. "Software defined cognitive networking: Supporting intelligent online video streaming" 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, 2018, pp. 1-4. doi: 10.1109/CCNC.2018.8319167.

[107] Megyesi, P., Botta, A., Aceto, G., Pescapè, A., Molnár, S., "Available Bandwidth Measurement in Software Defined Networks.", SAC 2016, April 04 - 08, 2016, Pisa, Italy. http://dx.doi.org/10.1145/2851613.2851727

[108] Noghani, K. A., M. Oguz Sunay, M., “Streaming Multicast Video over Software-Defined Networks”, IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems (MASS 2014), 28-30 Oct, 2014, Philadelphia, PA, USA, pp. 551 – 556. http://dx.doi.org/10.1109/MASS.2014.125

[109] Jarschel, M., Wamser, F., Hohn, T., Zinner, T., Tran-Gia, P., “SDN-based Application-Aware Networking on the Example of YouTube Video Streaming”, 2013 Second European Workshop on Software Defined Networks (EWSDN 2013), Oct. 10 - 11, 2013, Berlin, Germany, pp. 87-92. http://doi.ieeecomputersociety.org/10.1109/EWSDN.2013.21

[110] Salsano, S., Ventre, P. L., Prete, L., Siracusano, G., Gerola, M., Salvadori, E., "OSHI-Open Source Hybrid IP/SDN networking (and its emulation on Mininet and on distributed SDN testbeds).", 2014 Third European Workshop on Software Defined Networks (EWSDN), IEEE, 1-3 September 2014, Budapest, Hungary, pp. 13-18. http://dx.doi.org/10.1109/EWSDN.2014.38

[111] J. Huang, Q. Duan, Y. Zhao, Z. Zheng, W. Wang, "Multicast Routing for Multimedia Communications in the Internet of Things," IEEE Internet of Things Journal, vol.4, no. 1, Feb. 2017, pp 215-224. https://doi.org/10.1109/JIOT.2016.2642643

[112] Z. Quin, G. Denker, C. Giannelli, P. Bellavista, N. Venkatasubramanian, "A Software Defined Networking architecture for the Internet-of-Things," 2014

IEEE Network Operations and Management Symposium (NOMS), 5-9 May, 2014, Krakow, Poland, https://doi.org/10.1109/NOMS.2014.6838365

[113] N. Omnes, M. Bouillon, G. Fromentoux, O. Le Grand, "A programmable and virtualized network& IT infrastructure for the internet of things: How can NFV & SDN help for facing the upcoming challenges," 2015 18th International Conference on Intelligence in Next Generation Networks (ICIN), 17-19 Feb., 2015, Paris, France, https://doi.org/10.1109/ICIN.2015.7073808

[114] N. Bizanis, F. A. Kuipers, "SDN and Virtualization Solutions for the Internet of Things: A Survey," IEEE Access, vol. 4, 09 September, 2016, pp. 5591-5606, https://doi.org/10.1109/ACCESS.2016.2607786

[115] A. L. Valdivieso Caraguay, A. B. Peral, L. I. Barona López, L. J. García Villalba, "SDN: Evolution and Opportunities in the Development IoT Applications," International Journal of Distributed Sensor Networks, vol. 10, no. 5, January 1, 2014, Article ID 735142, 10 pages.
https://doi.org/10.1155/2014/735142

[116] K. Sood, S. Yu, Y. Xiang, "Software-Defined Wireless Networking Opportunities and Challenges for Internet-of-Things: A Review," IIEEE Internet of Things Journal, vol. 3, no. 4, August 2016, pp. 453-463.
https://doi.org/10.1109/JIOT.2015.2480421

[117] A. Pal, "Internet of Things: Making the Hype a Reality," IT Professional, vol. 17, no. 3, May-June 2015, pp. 2-4.
https://doi.org/10.1109/MITP.2015.36)

[118] M.Hassanalieragh, A. Page, T. Soyata, G. Sharma, M. Aktas, G. Mateos, B. Kantarci, S. Andreescu, "Health Monitoring and Management Using Internet-of-Things (IoT) Sensing with Cloud-Based Processing: Opportunities and Challenges," 2015 IEEE International Conference on Services Computing, June 27-July 2, 2015, New York, NY, USA.
https://doi.org/10.1109/SCC.2015.47

[119] L. Parra, S. Sendra, J. M. Jimenez, J. Lloret, "Multimedia sensors embedded in smartphones for ambient assisted living and e-health," Multimedia Tools and Applications, vol. 75, no. 21, November 2016, pp. 13271–13297, https://doi.org/10.1007/s11042-015-2745-8

[120] L. Tan, N. Wang, "Future Internet: The Internet of Things," 2010 3rd International Conference on Advanced Computer Theory and Engineer-

ing (ICACTE), 20-22 Aug., 2010, Chengdu, China.
https://doi.org/10.1109/ICACTE.2010.5579543

[121] J. Gubbia , R. Buyyab, S. Marusic, M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol. 29, no. 7, September 2013, pp. 1645-1660, September 2013.
https://doi.org/10.1016/j.future.2013.01.010

[122] S. O. Ajiboye, P. Birch, C. Chatwin, R. Young, "Hierarchical video surveillance architecture: a chassis for video big data analytics and exploration," Proc. SPIE 9407, Video Surveillance and Transportation Imaging Applications 2015, 94070K (4 March 2015), San Francisco, California, United States, https://doi.org/10.1117/12.2083937

[123] J. Lloret, P.V. Mauri, J.M. Jimenez, J.R. Diaz, "802.11g WLANs Design for Rural Environments Video-surveillance," ICDT '06. International Conference on Digital Telecommunications, 29-31 Aug., Cap Esterel, Côte d'Azur, France, 2006.
https://doi.org/10.1109/ICDT.2006.1

[124] H. Yang, J. Zhang, Y. Zhao, Y. Ji, J. Han, Y. Lin, S. Qiu, Y. Lee, Time-aware Software Defined Networking for OpenFlow-based Datacenter Optical Networks, Network Protocols and Algorithms, vol. 6, no. 4, pp. 77-91, 2014.

[125] M. Dramitinos, N. Zhang, M.. Kantor, J. Costa-Requena, I. Papafili, "Video Delivery over Next Generation Cellular Networks" Network and Service Management (CNSM), 2013 9th International Conference, IEEE, 14-18 Oct-2013, Zurich (Switzerland), pp. 386-393, doi> 10.1109/CNSM.2013.6727862t

[126] P. Gallo, K. Kosek-Szott, S. Szott and I. Tinnirello, "SDN@home: A method for controlling future wireless home networks," in IEEE Communications Magazine, vol. 54, no. 5, pp. 123-131, May 2016.

[127] Ginsberg L., Litkowski S., Previdi S., IS-IS Route Preference for Extended IP and IPv6 Reachability, RFC 7775, DOI: 10.17487/RFC7775, February 2016, https://www.rfc-editor.org/rfc/rfc7775.txt

[128] Rekhter Y., Li T., Hares S:, A Border Gateway Protocol 4 (BGP-4), RFC 4271.
DOI: 10.17487/RFC4271, January 2006, https://rfc-editor.org/rfc/rfc4271.txt

[129] Caria M., Das T., Jukan A., Divide and conquer: Partitioning OSPF networks with SDN, IFIP/IEEE International Symposium on Integrated Network Management (IM 2015), 11-15 May, Ottawa (ON), Canada, 2015.
https://doi.org/10.1109/INM.2015.7140324

[130] Rothenberg C. E., Nascimento M. R., Salvador M. R., Corrêa C. N. A., Cunha de Lucena S., Raszuk R., Revisiting routing control platforms with the eyes and muscles of software-defined networking, HotSDN '12 Proceedings of the first workshop on Hot topics in software defined networks, August 13 - 17, Helsinki (Finland), pp 13-18, 2012. https://doi.org/10.1145/2342441.2342445

[131] Zhu M., Cao J., Pang D., He Z., Xu M., "SDN-Based Routing for Efficient Message Propagation in VANET, In: Wireless Algorithms", Systems, and Applications (WASA 2015), Lecture Notes in Computer Science, vol. 9204, pp 788-797, https://doi.org/10.1007/978-3-319-21837-3_77

[132] Abbas, M.T., Muhammad, A.& Song, W, "SD-IoV: SDN enabled routing for internet of vehicles in road-aware approach" J Ambient Intell Human Comput vol. 11, pp. 1265–1280, 2020. https://doi.org/10.1007/s12652-019-01319-w

[133] A. L. Valdivieso Caraguay, J. A. Puente Fernández and L. J. García Villalba. Framework for optimized multimedia routing over software defined networks. Computer Networks. 2015; vol. 9, pp. 369-379. Doi: 10.1016/j.comnet.2015.09.013.

[134] M. Amiri, H. Al Osman, S. Shirmohammadi, and M. Abdallah. "SDN-based game-aware network management for cloud gaming". In 2015 IEEE International Workshop on Network and Systems Support for Games (IEEE NetGames 2015), Zagreb, 2015, pp. 1-6.

[135] Sendra S., Fernández P. A., Quilez M. A., Lloret J., "Minimizing packet loss by optimizing OSPF Gateway IP routing Protocols, Network Protocols and Algorithms", vol. 2, no. 4 (2010), https://doi.org/10.5296/npa.v2i4.547

[136] Rakheja P., Kaour P., Gupta A., Sharma A., "Performance Analysis of RIP, OSPF, IGRP and EIGRP Routing Protocols in a Network", International Journal of Computer Applications, vol. 48, no. 18, pp. 6-11,
https://doi.org/10.5120/7446-0401

[137] Barbancho J., León C., Molina J., Barbancho A., "SIR: A New Wireless Sensor Network Routing Protocol Based on Artificial Intelligence", In Advanced Web and Network Technologies, and Applications. APWeb 2006.

Lecture Notes in Computer Science (LNCS), vol. 3842, pp. 271-275. https://doi.org/10.1007/11610496_35

[138] Barbancho J., León C., Molina F. J., Barbancho A., Using Artificial Intelligence in Wireless Sensor Routing Protocols. Knowledge-Based Intelligent Information and Engineering Systems. (KES 2006). Lecture Notes in Computer Science, vol. 4251. Springer, pp. 475-482. https://doi.org/10.1007/11892960_58

[139] Arabshahi P., Gary A., Kassabalidis I., Das A., Narayanan S., Sharkawi M. El, Marks Ii , R. J. (2001), "Adaptive Routing in Wireless Communication Networks using Swarm Intelligence", AIAA 19th Annual Satellite Communications System Conference, Toulouse, France, April 17, 2001.

[140] Gunes M., Sorges U., Bouazizi I., "ARA-the ant-colony based routing algorithm for MANETs", International Conference on Parallel Processing Workshops, Vancouver, BC, Canada, 21-21 Aug, 2002. https://doi.org/10.1109/ICPPW.2002.1039715

[141] Ducatelle F., Di Caro G. A., Gambardella L. M., "Principles and applications of swarm intelligence for adaptive routing in telecommunications networks", Swarm Intelligence, September 2010, vol. 4, no. 3, pp. 173–198, https://doi.org/10.1007/s11721-010-0040-x

[142] Rajagopalan S., Shen C., "ANSI: A swarm intelligence-based unicast routing protocol for hybrid ad hoc networks", Journal of Systems Architecture, vol. 52, no. 8–9, August–September 2006, pp. 485-504. https://doi.org/10.1016/j.sysarc.2006.02.006

[143] RFC 3561 Ad hoc On-Demand Distance Vector (AODV) Routing, July 2003. Available online: https://www.rfc-editor.org/info/rfc3561 (accessed on 08 05 2018).

[144] Zungeru A. M., Ang L., Seng K. P., "Classical and swarm intelligence based routing protocols for wireless sensor networks: A survey and comparison", Journal of Network and Computer Applications, vol. 35, no. 5, September 2012, pp. 1508-1536, https://doi.org/10.1016/j.jnca.2012.03.004

[145] Karaboga D., Okdem S., Ozturk C., "Cluster based wireless sensor network routing using artificial bee colony algorithm", Wireless Networks, October 2012, Vol. 18, no. 7, pp. 847–860, https://doi.org/10.1007/s11276-012-0438-z

[146] H. A. A. Al-Rawi, M. A. Ng, and K.-L. A. Yau, “Application of reinforcement learning to routing in distributed wireless networks: a review”, Artificial Intelligence Review, 2015. vol. 43, no. 3, pp. 381-416.

[147] R. Desai, and BP. Patil, “Cooperative reinforcement learning approach for routing in ad hoc networks”. In proc. of the 2015 International Conference on Pervasive Computing (IEEE ICPC 2015), January 8-10, 2015, Pune (India). Pp. 1-5.

[148] J. Solanki, and A. Chauhan, “A Reinforcement Learning Network based Novel Adaptive Routing Algorithm for Wireless Ad-Hoc Network”. International Journal of Science Technology& Engineering., 2015, vol. 1, no. 12, pp. 135-142.

[149] MALKIN, Gary, et al., “RIP version 2. STD 56, RFC 2453”, November, 1998.

[150] Coltun R., Ferguson D., Moy J., OSPF for IPv6, RFC 5340, DOI: 10.17487/RFC5340, July 2008, https://rfc-editor.org/rfc/rfc5340.txt

[151] “OpenFlow Switch Specification version 1.0.0”, Available at:
https://www.opennetworking.org/wp-content/uploads/2013/04/
openflow-spec-v1.0.0.pdf [Last access: March 4, 2020]

[152] “OpenVSwitch”, Available at:
http://docs.openvswitch.org/en/latest/intro/what-is-ovs/ [Last access: March 4, 2020]

[153] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, S. Shenker, “Extending Networking into the Virtualization Layer,” HotNets-VIII, Oct. 22-23, 2009.

[154] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang, “Network virtualization in multi-tenant datacenters,” USENIX NSDI 2014.

[155] “Mininet”, Available at: http://mininet.org/ [Last access: March 4, 2020]

[156] R. L. S. de Oliveira, A. A. Shinoda, C. M. Schweitzer, P. L. Rodrigues, "Using Mininet for Emulation and Prototyping Software-Defined Networks", In ‘proceedings of the IEEE Communications and Computing (COLCOM 2014), June 4-6 , 2014, Bogota, Colombia, pp. 1-6.

[157] “Mininet-Wifi”, Available at: https://github.com/intrig-unicamp/mininet-wifi [Last access: March 4, 2020]

[158] “Quagga”, Available at: https://www.quagga.net/ [Last access: March 4, 2020]

[159] P. Jakma, and D. Lamparter, “Introduction to the quagga routing suite", IEEE Network 2014, vol. 28, no. 2, pp. 42-48.

[160] “Zebra”, Available at: http://www.zebra.org/ [Last access: March 4, 2020]

[161] BANNOUR, Fetia; SOUIHI, Sami; MELLOUK, Abdelhamid, “Distributed SDN control: Survey, taxonomy, and challenges”, IEEE Communications Surveys& Tutorials, vol. 20, no 1, pp. 333-354, 2018.

[162] “What is UML”. Available at: https://www.uml.org/what-is-uml.htm [Last access: July 28, 2020]

[163] ”The JSON Data Interchange Syntax”. Available at:
http://www.ecma-international.org/publications/files/
ECMA-ST/ECMA-404.pdf [Last access: May 5, 2020]

[164] “Introducing JSON”. Available at: https://www.json.org/json-en.html [Last access: May 5, 2020]

[165] “JSON in Java”. Available at:
https://mvnrepository.com/artifact/org.json/json [Last access: May 5, 2020]

[166] Ye T., Hema T. K., Kalyanaraman S., Vastola K. S, Yadav S., “Minimizing packet loss by optimizing OSPF weights using online simulation. Modeling, Analysis and Simulation of Computer Telecommunications Systems”. MASCOTS 2003. 11th IEEE/ACM International Symposium on, Orlando, FL, USA, 27 October 2003, https://doi.org/10.1109/MASCOT.2003.1240645.

[167] Cian O’Halloran, “Dynamic Adaptation of OSPF Interface Metrics based on Network Load”, 26th Irish Signals and Systems Conference (ISSC), Ireland. Jun 2015, https://doi.org/10.1109/ISSC.2015.7163767

[168] Mehmet ŞİMŞEK, Nurettin Doğan, Muhammet Ali Akcayol, “A New Packet Scheduling Algorithm for Real-Time Multimedia Streaming”, Network Protocols and Algorithms, vol. 9, No. 1-2 (2017). Pp. 28-47.
https://doi.org/10.5296/npa.v9i1-2.12410

[169] Ramon Sanchez-Iborra, Maria-Dolores Cano, Joan Garcia-Haro, “Revisiting VoIP QoE assessment methods: are they suitable for VoLTE?”, Network Protocols and Algorithms, vol. 8, no. 2 (2016). Pp. 39-57. https://doi.org/10.5296/npa.v8i2.9123

[170] P. Symes, "Digital Video Compression," Ed. New York, NY, USA: McGraw-Hill, 2004, 394 pp.

[171] A. Bovik, "Handbook of Image and Video Processing," San Diego, CA, USA: Academic Press, 2000, 1384 pp.

[172] Z. Wang, A.C. Bovik, H. R. Sheikh, E.P. Simoncelli, “Image Quality Assessment: From Error Visibility to Structural Similarity,” IEEE Transactions on Image Processing, vol. 13, no. 4, April 2004, pp. 600-612, https://doi.org/10.1109/TIP.2003.819861.

[173] Zhou Wang and A. C. Bovik, "A universal image quality index," in IEEE Signal Processing Letters, vol. 9, no. 3, pp. 81-84, March 2002, doi: 10.1109/97.995823.

[174] Kayri, Murat, “Predictive Abilities of Bayesian Regularization and Levenberg–Marquardt Algorithms in Artificial Neural Networks: A Comparative Empirical Study on Social Data”, Mathematical and Computational Applications, vol. 21, pp. 1-11, 2016.

[175] Hirschen K, Schafer M, “Bayesian regularization neural networks for optimizing fluid flow processes”, Computer Methods in Applied Mechanics and Engineering, vol. 195, no. 7, pp. 481–500, 2006.

[176] Martin HT, Menhaj MB, “Training feed forward networks with the Marquardt algorithm”, IEEE Trans Neural Networks, vol. 5, no. 6, pp. 989–993, 1994.

[177] Ozgur Kisi, Erdal Uncuoğlu, “Comparison of three back-propagation training algorithms for two case studies”, Indian Journal of Engineering and Materials Sciences, vol. 12, no. 5, 2005.

[178] S. Mohanty, M. K. Jha, A. Kumar, K. P. Sudheer, “Artificial neural network modeling for groundwater level forecasting in a river island of eastern India”, Water Resources Management, vol. 24, no. 9, pp. 1845–1865, 2010.

[179] H. P. Gavin, “The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems”, Department of Civil and Environmental Engineering, Duke University, pp. 1–17, 2013.

[180] Yi Lin, “Support Vector Machines and the Bayes Rule in Classification”, Data Mining and Knowledge Discovery, vol. 6, no. 3, pp. 259-275, 2002.

[181] Steven N Goodman, "Introduction to Bayesian methods I: measuring the strength of evidence“, Clinical Trials, vol. 2, no. 4, pp. 282–290, 2005.

[182] Foresee and Hagan, 1997 F.D. Foresee, “M.T. HaganGauss–Newton approximation to Bayesian learning”, Proceeding of the 1997 International Joint Conference on Neural Networks, pp. 1930-1935, Houston, 1997.

[183] Durden F., Winkler D., “Bayesian Regularization of Neural Networks”, Artificial Neural Networks, Methods in Molecular Biology™, vol. 458, pp. 23-42, 2008.

[184] A. Rego, A. Canovas, J. M. Jiménez and J. Lloret, "An Intelligent System for Video Surveillance in IoT Environments," IEEE Access, vol. 6, pp. 31580-31598, 2018.

[185] S. Islam, N. Muslim and J. W. Atwood, “A Survey on Multicasting in Software-Defined Networking”, IEEE Communications Surveys& Tutorials, vol. 20, pp. 355-387, 2018. doi: 10.1109/COMST.2017.2776213

[186] L. García, J. M. Jiménez, M. Taha and J. Lloret, "Video artifact evaluation based on qos and objective qoe parameters," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 1048-1054, doi: 10.1109/ICACCI.2017.8125980.

[187] M. Taha, L. Garcia, J. M. Jimenez and J. Lloret, "SDN-based throughput allocation in wireless networks for heterogeneous adaptive video streaming applications," 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, 2017, pp. 963-968, doi: 10.1109/IWCMC.2017.7986416.

[188] Winarno Sugeng, Jazi Eko Istiyanto, Khabib Mustofa and Ahmad Ashari, “The Impact of QoS Changes towards Network Performance”, International Journal of Computer Networks and Communications Security, vol. 3, no. 2, pp. 48-53, 2015.

[189] Aruba 2930F Data Sheet. Available at: https://www.arubanetworks.com/assets/ds/DS_2930FSwitchSeries.pdf [Last access: October 7, 2020]

[190] Big Buck Bunny website. Available at: http://www.bigbuckbunny.org [Last access: October 7, 2020]

[191] VLC media player website. Available at: http://www.videolan.org [Last access: October 7, 2020]

[192] Wireshark website. Available at: https://www.wireshark.org/ [Last access: October 7, 2020]

[193] Marín, J., Rocher, J., Parra, L., Sendra, S., Lloret, J., and Mauri, P. V, "Autonomous WSN for Lawns Monitoring in Smart Cities", 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), Hammamet, Tunisia, Oct 30, 2017 - Nov 3, 2017, pp. 501-508.

[194] Parra, L., Rocher, J., García, L., Lloret, J., Tomás, J., Romero, O., Rodilla, M., Falco, S., Sebastiá,M.T., Mengual, J., González, J.A., Roig B, "Design of a WSN for smart irrigation in citrus plots with fault-tolerance and energy-saving algorithms", Network Protocols and Algorithms, vol. 10, pp. 95-115, 2018.

[195] Lloret, J.; Tomás Gironés, J., Canovas Solbes, A., Parra-Boronat, L., "An Integrated IoT Architecture for Smart Metering", IEEE Communications Magazine. vol. 54, pp. 50-57, 2016.

[196] Sendra, S., Parra, L., Lloret, J., and Jiménez, J. M., "Oceanographic multisensor buoy based on low cost sensors for Posidonia meadows monitoring in Mediterranean Sea", Journal of Sensors, vol. 2015, 23 pages, 2015. https://doi.org/10.1155/2015/920168

[197] S. Sandra, J. Lloret, M. Garcia, J.F. Toledo, "Power saving and energy optimization techniques for Wireless Sensor Networks", Journal of Communications, vol. 6, no. 6, pp. 439-459, 2011.

[198] W. R. Heinzelman, A. Chandrakasan, A., H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks", In Proceedings of the IEEE 33rd annual Hawaii international conference on System Sciences, Maui, Hawaii, January 4-7, 2000.

[199] J.P. Kaps, B. Sunar, "Energy Comparison of AES and SHA-1 for Ubiquitous Computing", In Proceedings of the EUC 2006 Workshops: NCUS, SecUbiq, USN, TRUST, ESO, and MSA, Seoul, South Korea, August 1-4, 2006.