University of Genoa

Ph.D. in Science and Technology for Electronic and Telecommunication Engineering

Doctoral Thesis

Advanced SDN-Based QoS and Security Solutions for Heterogeneous Networks

Candidates:

Luca Boero

Advisor:

Prof. Mario Marchese

Academic Year 2018/2019

Yesterday is history,

future is a mystery,

but today is a gift.

That's why it's called present.

Master Oogway

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Mario Marchese, for his excellent guidance and for trusting in me since the beginning.

I am sincerely grateful to my colleague Fabio Patrone and all the people working in the *Satellite Communications and Networking Laboratory* for providing me with an excellent atmosphere for doing research. My work would not have been possible without their help.

Moreover, I would like to thank my Relatives and my girlfriend's Family for their support, trust, and love. Their caring help during these years allowed me to pursue my projects and realize my dreams.

Last but not least, I would like to thank my girlfriend Chiara for her everyday support and love during difficult moments.

Luca

Contents

1 Introduction **11**

2 Software Defined Networking **16**

2.1 The SDN Controller 21

2.2 The OpenFlow Protocol 22

3 QoS in Traditional and SDN Networks **26**

3.1 Definition of Quality of Service 26

3.2 QoS Classes, Applications and Metrics 28

3.3 Approaches to QoS Management 30

3.4 QoS Management in OpenFlow 32

3.5 Related works 34

3.6 Motivations 36

3.7 Possible solutions 39

3.7.1 General Idea 39

3.7.2 Implementation: BeaQoS 42

3.8 Re-Routing Strategies Analysis 45

3.8.1 Deadline Management Scenario 45

3.8.2 Queue Balancing Scenario 50
3.9 Considerations 56
3.9.1 Scaling Performances 56
3.9.2 Switch Coordination 57
3.9.3 Considerations on Queue Balancing Scenario 57
3.10 Conclusion about Support of Quality of Service in SDN 58

4 SDN in Satellite Environment 60
4.1 State of the Art for SDN/NFV Enabled Satellite Networks . . 63
4.2 Open Challenges 67
4.3 Proposed Solutions 71
4.4 The role of SDN in the 5G Satellite Communications 76
4.5 SDN-based Satellite Terrestrial Network 79
4.6 Time estimation model 84
4.7 Results and Final considerations 89
4.8 Conclusion about the role of SDN in the Satellite environment 93

5 The Problem of Security 96
5.1 State of the art 98
5.1.1 Machine Learning-based Classifiers 98
5.1.2 Intrusion Detection Systems classification 101
5.2 Statistical Fingerprint-IDS 106
5.2.1 Key Ideas 106
5.2.2 SF-IDS Architecture 107
5.2.3 SF-IDS Packet Analyzer 109
5.2.4 Statistical Fingerprint: the feature vector 110

5.2.5 SF-IDS Classifiers 111
5.3 Used traffic and performance parameters 113
5.3.1 Used Malware and Normal Traffic 113
5.3.2 Performance Evaluation Parameters 114
5.4 Performance evaluation 117
5.4.1 Tools 117
5.4.2 Evaluation of Single Classifiers 117
5.4.3 Single Malware affected flows performance 119
5.4.4 Classifiers Acting in Parallel 120
5.5 Considerations 123

6 SDN IDS Implementation 126
6.1 Introduction 127
6.2 State of The Art 127
6.3 SDN Controller Employed within an IDS 129
6.3.1 General Description of Ryu 129
6.3.2 Flow Structure 130
6.3.3 Structure of the "Stats_Manager" Application 132
6.4 Used Traffic 136
6.5 Testbed and Scenario Setup 138
6.5.1 The network 138
6.5.2 Traffic emulation 140
6.5.3 The Classifier 141
6.6 Performance evaluation and comments 142

7 Conclusions 146

List of Figures

2.1 Difference between traditional and SDN network. 18
2.2 Routing in traditional vs. SDN network. 19

3.1 Congestion at one of the queue. 40
3.2 Action of re-routing of some flows. 40
3.3 Matched Flows with $H = 100$ 48
3.4 Matched Flows with $H = 250$ 49
3.5 Background Loss with $H = 100$ 50
3.6 Background Loss with $H = 250$ 51
3.7 Queue balancing performances with 100 flows. 54
3.8 Queue balancing performances with 125 flows. 55
3.9 Queue balancing performances with 150 flows. 55
3.10 Timing Performances in Queue Balancing Scenario. 58

4.1 Architectural Framework 71
4.2 Road-map for an SDN/NFV-enabled satellite network 74
4.3 Considered terrestrial-satellite network 79
4.4 Polar LEO satellite constellation 81
4.5 LEO satellite constellation ISL model 82

4.6 SDN Planes and functionalities scheme 83
4.7 Minimum and maximum $\overline{T}$ 91

5.1 SF-IDS overall architecture. 108
5.2 Block diagram of the Packet Analyzer. 110

6.1 Ryu scheme 129
6.2 General picture of how the app interacts with Ryu 132
6.3 Flow Stats Reply 134
6.4 Classification Process 135
6.5 Network topology 139
6.6 ROC Diagram of Simulation 1 143
6.7 ROC Diagram of Simulation 2 144

List of Tables

2.1 SDN controllers overview 23

3.1 Possible example of Service Level Specification. 28
3.2 ITU-T Y-1541 Qos Classes. 29
3.3 1-queue and 3-queues configurations performance 36
3.4 Fixed and variable service rate performance 37
3.5 Traffic classes and their deadline requirements 38
3.6 BeaQoS Statistics compared with OpenFlow 1.0 statistics. . . 44
3.7 Queue configurations. 47

4.1 Matching between challenges and Architectural Elements . . . 73
4.2 List of defined variables 89
4.3 Set of initial parameters used for the experiments 91
4.4 P_L and S_L needed for the complete coverage of the Earth . . . 92

5.1 Misuse vs Statistical Fingerprint IDS 102
5.2 Used features for each flow as Statistical Fingerprint. 112
5.3 Used malware and normal traffic. 115
5.4 Evaluation Parameters. 116

5.5 Acc, TP, FN, TN, FP, and CI varying the applied Classifier 118
5.6 Acc, TP, FN, TN, FP, and CI for J48 120
5.7 Acc, TP, FN, TN, FP, and CI for PART 121
5.8 Acc, TP, FN, TN, FP, and CI for Random Forest 122
5.9 Acc, TP, FN, TN, FP, and CI for the three DMs 123

6.1 Flow Dictionary Structure 130
6.2 Results of the Simulations 142

Chapter 1

Introduction

Software Defined Networking (SDN) is revolutionizing the networking industry by enabling programmability, easier management and faster innovation. These benefits are made possible by its centralized control plane architecture, which allows the network to be programmed by the application and controlled from one central entity. The SDN architecture is composed of both switches/routers and a central controller (SDN controller). The SDN device processes and delivers packets according to rules stored in its flow table (forwarding state), whereas the SDN controller configures the forwarding state of each switch using a standard way: OpenFlow (OF). The controller is also responsible to build the virtual topology representing the physical one. Virtual topology is used by application modules that run on top of the SDN controller to implement different control logics and network functions (e.g., routing, traffic engineering, firewall state).

Currently Quality of Service (QoS) management in OpenFlow is limited: in each OF switch one or more queues can be configured for each outgoing interface and used to map flow entries on them. Flow entries assigned to a specific queue will be treated according to the queue's configuration in terms of service rate, but queue configuration takes place outside the OpenFlow protocol [1]. Without a proper rate assignment, it is difficult to guarantee QoS requirements to traffic. A possible solution to limit performance degradation involves the re-routing of the flow undergoing the violation of QoS constraints on a less congested path or queue.

Satellite communication networks have been evolving from standalone networks with ad-hoc infrastructures to possibly interconnected portions of a wider Future Internet architecture. Experts belonging to the fifth-generation (5G) standardization committees are considering satellites as a technology to integrate in the 5G environment. SDN is one of the paradigms of the next generation of mobile and fixed communications. It can be employed to perform different control functionalities, such as routing, because it allows traffic flow identification based on different parameters and traffic flow management in a centralized way. The problem is to individuate a possible SDN-based satellite - terrestrial network architecture and understand if the classical way of management of this type of network (using OpenFlow) is feasible.

Many infrastructures and services are based on the use of Internet. The number of people that has access to an Internet connection is growing rapidly. These facts lead, on one hand, to new possible threats and attacks used by cyber criminals, and, on the other hand, to an increased complexity in the management. It is crucial to design systems able to prevent and tackle cyber attacks such as Intrusion Detection Systems (IDS) that can alert when someone or something is trying or has tried to compromise information systems through malicious actions. In the same time, big efforts are provided in order to get tools that can make easier network management. The SDN paradigm has been designed with this aim and allows network administrators to manage networks by abstracting functionalities through the separation of data and control planes.

In this thesis we try to study how SDN can be employed in order to support Quality of Service and how the support of this functionality is fundamental for today networks. Considering, not only the present networks, but also the next generation ones, the importance of the SDN paradigm become manifest as the use of satellite networks, which can be useful considering their broadcasting capabilities. For these reasons, this research focuses its attention on satellite - terrestrial networks and in particular on the use of SDN inside this environment. As mentioned before, the growing of the information technologies has pave the way for new possible threats. This research study tries to cover also this problem considering how SDN can be employed for the detection of past and future malware inside networks.

The outcomes of this thesis are manifold, and in particular can be expressed as:

Theoretical Develop a theoretical framework able to map topology, QoS requirements and traffic descriptors into actions that must be taken in order to provide better Quality of Service and guarantee certain security level inside a network. Describe the next generation satellite networks trying to understand what are the problems of the current generation and provide possible solutions and a possible road-map in order to effectively implement the SDN paradigm in a satellite - terrestrial network.

Algorithm design Design of algorithms and routines that, given the status of the network, can generate network-level directives (i.e. flow modifications in OpenFlow) for the devices in the network in order to manage the overall Quality of Service but also can stop malicious traffic that is traversing the network.

Implementation Implementation of a QoS module and a security module inside an SDN controller: the controller will implement the solutions carried out in the theoretical and algorithm design objectives. The implemented controller will be tested through simulated/emulated scenarios in order to evaluate performance issues and scalability concerns.

The studies made during my Ph.D. on these problems allow me to create two SDN controller prototypes module, which are dedicated to the quality of service and to network security respectively. The first module is responsible for collecting traffic statistics from the underling network, monitoring the queues of SDN switches and re-route flows in case of traffic overloading. In the second one, thanks to the collection of statistics developed for the first module and with simple computations, the security aware controller module can distinguish in real time if a traffic flow is affected by malware or not. Another fundamental part of my research activity is dedicated to the use of SDN technology in the satellite environment. One of the aims of my research is to introduce an SDN-based terrestrial satellite network architecture and to estimate the mean time to deliver the data of a new traffic flow from the source to the destination including the time required to transfer SDN control actions. The practical effect is to identify the maximum performance than can be expected in this type of network.

Software Defined Networking is a matter of high interest for the academic research area since it can provide answers to one of the main problem of the networking: portability. In the traditional approach to networking, most network functionalities are implemented in routers and switches with dedicated hardware but this approach is characterized by slow evolution of network functionalities. As the traditional networking paradigm has a rather static nature and now there is a widespread adoption of server virtualization and a pressure for network organizations to be more efficient and agile in network management, the necessity of a new software-oriented approach starts to arise. One of the main solutions that fulfill the aforementioned need is Software Defined Networking. SDN is highly beneficial for what concerns the QoS, Security and satellite worlds.

The thesis is organized as follows: Chapter 2 describes the SDN paradigm and its main components. Chapter 3 introduces the concepts of Quality of Service and how it is handled inside traditional and SDN-based networks. Chapter 4 shows the use of SDN inside the satellite environments and tries to understand limitations and solutions that can be applied to this environment. Chapter 5 describes the problem of network security with particular interest in the use of intrusion detection systems. Chapter 6 describes the use of an SDN architecture as an intrusion detection systems able to identify the presence of malware inside a traffic flow. Finally Chapter 7 concludes the thesis.

Chapter 2

Software Defined Networking

In the traditional approach to networking, most network functionality is implemented in a dedicated appliance such as switch, router, application delivery controller. In addition, within the dedicated appliance, most of the functionality is implemented in dedicated hardware such as an ASIC (Application Specific Integrated Circuit).
This kind of approach is characterized by slow evolution of network functionalities, which are by the way under the control of the provider of the device. The widespread adoption of server virtualization and the consequent need to move virtual machines dynamically between servers lead to increasing pressure for network organizations to be more efficient and agile in network management. As the traditional networking paradigm has a rather static nature, the need for a new software-oriented approach started to arise.

One of the main innovations that the software-based approach can introduce is the concept of abstractions applied to the control plane. The abstractions in the data plane are already well established: the use of the protocol stack is well standardized in any networking application. We need to define some abstractions useful to create reusable components also for the control plane [2]. The main tasks that the control plane must take care of are:

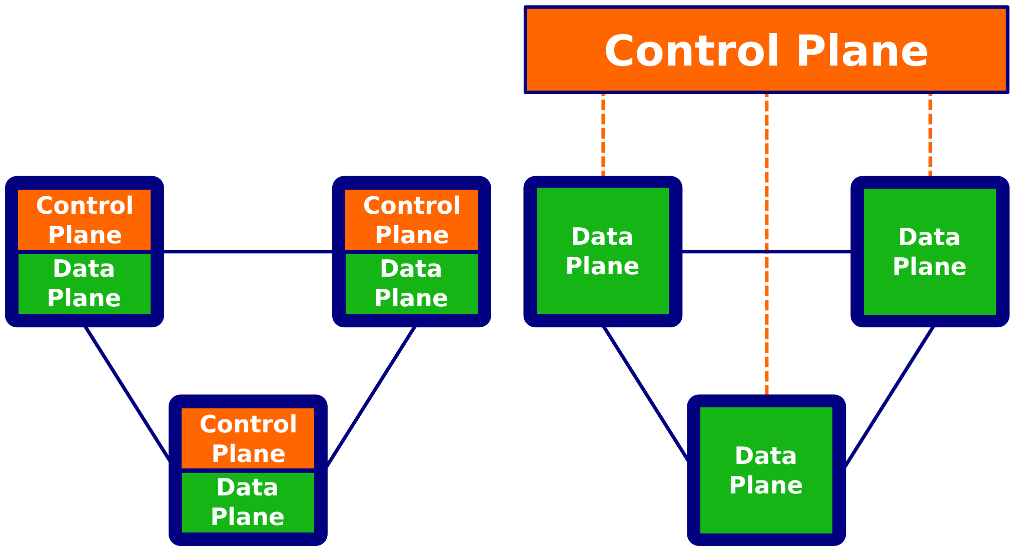
- Figuring out the topology of the network
- Computing how to accomplish its goal on the given topology
- Configuring the forwarding state

Till now, each new control protocol must solve all three issues. In order to create components that can be reused, it is possible to abstract the two main tasks related to the topology information and the forwarding state configuration. These two issues are identified by:

Global Network View It provides information about all the topology architecture related to the considered network. This abstraction is intended to translate the real network structure into a graph that the controller can analyze in order to compute the forwarding model. The global network view is provided by the Network Operating System (NOS), which is an implementation of this abstraction lying above the data plane.

Forwarding Model It provides a standard way of defining the forwarding state inside the network nodes. The implementation of this abstraction consists in installing simple forwarding rules inside the SDN switches that are part of the network. The nodes are then supposed to follow these rules in order to forward the packets.

The main innovation brought by Software Defined Networking is the decoupling of control and data plane [3]. In legacy networks, both control an data plane are managed by the same entity, for example the router. This concept creates a heavy burden for the device which has to take care of the network. In Software Defined Networking the two tasks are assigned to different entities: the controller, which is the part of the network dedicated to compute the forwarding state and the switch, which is the node devoted to packet forwarding based on the local forwarding state. This difference is highlighted in Figure 2.1.



(a) (b)

Figure 2.1: Difference between a traditional network 2.1(a) and a SDN network 2.1(b).

The main elements constituting a Software Defined Network are [4]:

- **Control Program** - It has to decide forwarding policy by knowledge of virtual global topology

- **Network OS** - It generates the virtual topology from the physical network

- **Routers/Switches** - They implement specific rules to forward packets

The Software Defined Networking architecture has the aim of leaving the intelligence outside the data plane, in order to make the forwarding mechanisms simpler and quicker. The control program can therefore be implemented inside a supervising Software Defined Networking controller together with the Network Operative System (Figure 2.2).

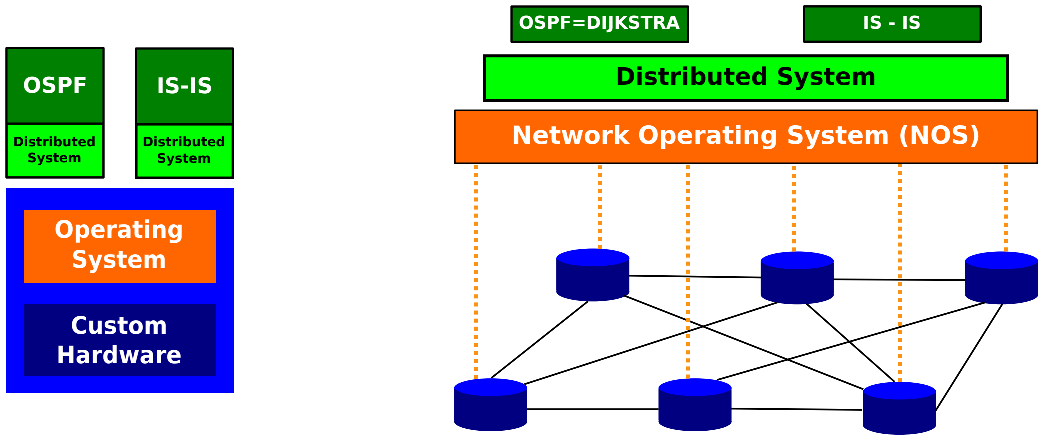


Figure 2.2: Difference between routing in a traditional network and a SDN network.

The switches, on the other hand, send packets on the basis of simple rules, installed inside the node by the above standing controller. These rules are defined by *match-action* couples: a packet whose header corresponds to certain parameters (a certain *match*) must be treated according to a specific policy (the *action* to be taken). Controller and switch interact together using the forwarding model, whose main implementation is the OpenFlow Protocol [1].

The main opportunities that Software Defined Networking can address are the support of dynamic movement and allocation of network resources, the scalability of network functionalities and the reduction of network complexity.
Software Defined Networking allows also to perform traffic engineering with an end-to-end view of the network and to apply more effective security functionalities [3].

The Open Networking Fundation (ONF) is the group that is most associated with the development and standardization of SDN . According to the ONF, Software Defined Networking is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications [5].
The main characteristics of the Software Defined Networking architecture are:

- *Programmability* - Network control is directly programmable because it is decoupled from forwarding functions
- *Agility* - Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs
- *Central Management* - Network intelligence is (logically) centralized in software-based Software Defined Networking controllers that maintain a global view of the network, which appears to applications and policy

engines as a single, logical switch

- *Programmatical Configuration* - Software Defined Networking lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated Software Defined Networking programs, which they can write themselves because the programs do not depend on proprietary software
- *Use of open standard* - When implemented through open standards, Software Defined Networking simplifies network design and operation because instructions are provided by Software Defined Networking controllers instead of multiple, vendor-specific devices and protocols

2.1 The SDN Controller

The decoupled system has been compared to an operating system [6], in which the controller provides a programmatic interface to the network, and it can be used to implement management tasks and offer new functionalities. This abstraction assumes the control is centralized and applications are written as if the network is a single system. It enables the Software Defined Networking model to be applied over a wide range of applications and heterogeneous network technologies and physical media such as wireless, wired and optical networks.

The Software Defined Networking controller has a double functionality: it needs to interact with the underlying network switches (*Southbound Communication*), but it also has to interface with high-level applications (*Northbound Communication*).

While controller-switch interaction is fairly well defined in protocols such as OpenFlow, there is no standard for interactions between controllers and network services or applications. One possible explanation is that the northbound interface is defined entirely in software, while controller-switch interactions must enable hardware implementation.

If we think of the controller as a “network operating system“, then there should be a clearly defined interface by which applications can access the underlying hardware (switches), co-exist and interact with other applications, and utilize system services (e.g., topology discovery, forwarding), without requiring the application developer to know the implementation details of the controller. While there are several controllers that exist, their application interfaces are still in the early stages and independent from each other. A global overview of the current controller implementations can be found in Table 2.1 [7].

2.2 The OpenFlow Protocol

OpenFlow [1] is a communication protocol that gives access to the forwarding plane of a switch or router over the network. It takes care of the signalling between the controller and the switches inside a Software Defined Networking network.

OpenFlow enables remote controllers to establish the path of packets that are travelling in the network, using network switches which are compliant to the standard. OpenFlow in fact makes available a tool, the *flow table*, which allows not only to perform routing operations, but also other functionalities,

Controller	**Implementation**	**Overview**
POX	Python	General, open-source SDN controller.
NOX	Python/C++	The first OpenFlow controller.
MUL	C	OpenFlow controller with a multi-threaded infrastructure that supports a multi-level north-bound interface for applications.
Maestro	Java	A NOS that provides interfaces for implementing modular network control applications.
Trema	Ruby/C	A framework for developing OpenFlow controllers.
Beacon	Java	A cross-platform, modular, OpenFlow controller that supports event-based and threaded operations.
Jaxon	Java	An OpenFlow controller based on NOX.
Helios	C	An extensible OpenFlow controller that provides a programmatic shell for performing integrated experiments.
Floodlight	Java	A controller based on Beacon, that works with physical- and virtual- OpenFlow switches.
SNAC	C++	An OpenFlow controller based on NOX, which manages the network, configures devices, and monitors events.
Ryu	Python	An SDN operating system that provides logically centralized control and APIs to create new network management and control applications.
NodeFlow	JavaScript	An OpenFlow controller for Node.JS
ovs-controller	C	A simple OpenFlow controller implementation with OpenvSwitch for managing remote switches through the OF protocol.
Flowvisor	C	Special purpose controller implementation.
RouteFlow	C++	Special purpose controller implementation.
ONOS	Java	Open Network Operating System (ONOS) is an (SDN) OS for service providers.

Table 2.1: SDN controllers overview

such as implementing firewall, making NAT (Network Address Translation) or implementing Quality of Service policies.

OpenFlow provides an open standard to program the flow table in different switches and routers. With this tool a network administrator can easily manage a network, divide flows or perform other actions without the need of hardware-specific tools but only using the OpenFlow primitives [8].

OpenFlow Protocol supports three message types: controller-to-switch, asynchronous and symmetric, each with multiple subtypes. The first type is used by the controller to manage or inspect the state of the switch. The second type of message is used by OpenFlow switch for updating the information the controller has about the network and the switch state. Symmetric messages are sent either by the switch or the controller without solicitation. There are different types of controller-to-switch messages:

- *Feature*: it is sent by the controller upon TLS (Transport Layer Security) session establishment. The switch replies to this message with a *features reply* that specifies the capabilities supported by the switch.

- *Modify-state*: this type of message is sent by the controller to manage states on switches. Its primary purpose is to add/remove and modify flows in the switch's flow table.

- *Read-state*: this message is used by the controller to collect statistics from the switch's flow table, port and individual flow entries.

- *Barrier*: this request/reply message is used by the controller to receive notifications for completed operations.

The asynchronous messages are sent by a switch without the controller's solicitation. The main type of asynchronous messages are:

- *Packet-in*: for all packets that do not have a matching flow entry in the flow table a packet-in event is sent to the controller.

- *Flow-removed*: this is a message sent to the controller if and only if the controller in the modify-state event has set the corresponding flag.
- *Errors*: the switch is able to notify the controller of problems using error messages.

The last type of messages is the symmetric ones:

- *Hello*: they are exchanged between the switch and the controller upon connection start-up.
- *Echo*: echo request/reply messages can be sent form either the switch or the controller and must return an echo reply. They can be used to indicate the latency, bandwidth and/or liveness of a controller-switch connection.
- *Vendor*: this type of message provides a standard way for OpenFlow switches to offer additional functionalities within the OpenFlow message type space.

Chapter 3

Quality of Service in Traditional and SDN Networks

3.1 Definition of Quality of Service

Concerning the network viewpoint, QoS is the ability of a network element to have some level of assurance that its traffic and service requirements can be satisfied. QoS manages bandwidth according to application demands and network management settings.

The term *QoS* is used in different meanings, ranging from the users' perception of the service to a set of connection parameters necessary to achieve particular service quality. The QoS meaning changes depending on the application field and on the scientific scope. Three types of Quality of Service have been defined [9]:

- **Intrinsic QoS** - it is directly provided by the network itself and may be described in terms of objective parameters, such as loss and delay.

- **Perceived QoS** - it is the quality perceived by the users; it heavily depends on the network performance but it is measured by the average opinion of the users. *Mean Opinion Score* (MOS) methods are often used to perform the measure of the quality. The MOS rating is the arithmetic mean of all the individual user scores and can range from 1 (worst) to 5 (best) [10]. Even if there is a strict connection with the objective metrics provided by the network, the user does not necessarily perceived an increase (or decrease) in performance, in correspondence of an intrinsic QoS variation.

- **Assessed QoS** - it is referred to the will of a user to keep on using a specific service. It is related to Perceived QoS (P-QoS) and also depends on the pricing mechanism, level of assistance of the provider and other marketing and commercial aspects. For example, a performance decrease may be surely tolerated by a user if a service is free, but the same decrease will raise criticism if the user is paying for it.

At the moment, most of the QoS provision is offered in terms of intrinsic QoS (objective parameters) by using a *Service Level Specification* (SLS) which is a set of parameters and their values which together define the service offered to a traffic [11]. SLS is a separated technical part of a negotiated agreement between a customer and the service provider on level of service characteristics and the associated set of metrics, which is the commonly adopted definition of a *Service Level Agreement* (SLA) [12, 13]. A possible example of Service Level Specification is shown in Table 3.1.

Concerning device and network capabilities, the SLA may also indicate the access technology that the customer can use, for example Ethernet, Wire-

Service Level Specification	**Range**
Connection type	CBR / VBR / Best-effort
Scope	End-to-end
Connection Identification	Identifier or sequence of identifiers
Traffic description	Packet rate / Bucket size / Max burst size
Performance guarantees	Packet loss / Packet transfer delay / Jitter
Multi Level Precedence	Not applied / Routine / Immediate / Flash

Table 3.1: Possible example of Service Level Specification.

less LANs, GSM, UMTS, Ad-hoc Network, GPRS and Satellite Access. Even if the SLS does not change, SLA should also consider these aspects that have a relevant impact on the customer choices.

3.2 QoS Classes, Applications and Metrics

Many applications nowadays need QoS. Some of them are telemedicine, tele-control (remote control of robots in hazardous environments, remote sensors and systems for tele-manipulation), tele-learning, telephony, video-conferences, online gaming, multimedia streaming and applications for emergencies and security. Each application, having very different characteristics, needs a specific degree of service, defined at the application layer.
Several standardization bodies have tried to define service categories, also called *QoS Classes*, to be intended at application layer. ITU-T suggests a definition of QoS classes [14] for the IP world which is summarized in Table 3.2.

A further step is to associate objective QoS requirements to the QoS

QoS Class	**Characteristics**
0	Real-time, jitter sensitive, highly interactive
1	Real-time, jitter sensitive, interactive
2	Transaction data, highly interactive
3	Transaction data, interactive
4	Low loss only (short transactions, bulk data, video streaming)
5	Traditional applications of default IP networks

Table 3.2: ITU-T Y-1541 Qos Classes.

traffic classes generically defined above. Concerning the IP environment, the QoS objective metrics mostly used [15] are as follows:

- **IPLR** - IP Packet Loss Ratio
- **IPTD** - IP Packet Transfer Delay
- **IPDV** - IP Packet Delay Variation (known as Jitter)
- **IPER** - IP Packet Error Ratio

Another metric often considered is the *skew*, which is the average value of the difference of the delays measured by packets belonging to different media such as voice and video within a video-conference service. In this case, if the skew is large, there is no synchronization between voice and video with the resulting effect of a bad dubbing.

3.3 Approaches to QoS Management

QoS management techniques are needed in order to offer the necessary tools to guarantee specific QoS requirements. A possible classification of QoS management functions is described in the following [10].

Over Provisioning It consists of purchasing an oversupply of bandwidth to solve the challenges. This approach ignores not only bandwidth optimization but also possible future trends and requirements of new services. It cannot be classified exactly as a solution.

Flow Identification The identification of packets so that they may receive a different treatment within the network is fundamental to guarantee QoS. Different technology show different method to classify packet: Flow Label and Traffic Class in IPv6, ToS and tuple [IP Source Address, IP Destination Address, Protocol, Transport Source Port, Transport Destination Port] in IPv4, VPI/VCI in ATM, Label Value in MPLS.

Call Admission Control An accurate resource reservation to guarantee that traffic flows receive the correct service is strictly needed. The acceptance or rejection of a new connection is performed subject to a check about the availability of network resources in consequence of specific requirements. After that, if enough resources are available, they are reserved.

Traffic Control (Shaping) Shaping policies limit flows to their committed rates in order for the flows to be conform with their traffic descriptors. If connections exceed their bandwidth consumption specifications, the network, which has dimensioned resources in strict dependence on the declarations, cannot guarantee any specified QoS requirement. Two common methods used in literature to shape traffic are Leaky Bucket and Token Bucket [16].

Scheduling Packet scheduling specifies the service policy of a queue of a node. In practice, scheduling decides the order that is used to pick the packets out of the queue and to transmit them over the channel. The main problem arises from the impossibility of assigning the committed bandwidth to a specific flow at each time instant. A clear and complete revision of the most interesting schedulers is reported in [17].

Queue Management Scheduling is often linked to queue management schemes. They are used, for example, to establish the dropping strategy when the buffer is full. Possible policies include: Tail Drop (discarding the packet arrived last), Front Drop (eliminating the first packet in the queue), Random Drop (selecting randomly the packet to discharge) or dynamic schemes.

Flow Control It is the process of managing the rate of data transmission between two nodes to prevent a fast sender from outrunning a slow receiver. It provides a mechanism for the receiver to control the transmission speed, so that the destination node is not overwhelmed with incoming data. Generally, flow control is implemented end-to-end at the transport layer. Even if it may help avoiding network saturation, it cannot guarantee a specific QoS requirement, if used alone.

3.4 QoS Management in OpenFlow

As previously mentioned, Software Defined Networking (SDN) is revolutionizing the networking industry by enabling programmability, easier management and faster innovation [18, 19].

The SDN architecture is composed both of SDN enabled devices and of a central controller (SDN controller). An SDN device processes and delivers packets according to the rules stored in its flow table (forwarding state), whereas the SDN controller configures the forwarding state of each SDN device by using a standard protocol called OpenFlow (OF) [19]. The SDN controller is responsible also to build the virtual topology representing the physical one. The virtual topology is used by application modules that run on top of the SDN controller to implement different control logics and network functions (e.g., routing, traffic engineering, firewall actions).

Currently the Quality of Service (QoS) management in OF is quite limited: in each OF switch one or more queues can be configured for each outgoing interface and used to map flow entries on them. Flow entries mapped to a specific queue will be treated according to the queue's configuration in terms of service rate, but the queue's configuration *takes place outside the OF protocol*. For example, the queue's service rate cannot be modified by OF.

Supposing that a flow is traversing a chain of queues from the source to the destination node, and the flow data rate increases, a possible consequence is that queues increase their occupancy, and a bottleneck may be generated with consequent network congestion. The impossibility to change the bottleneck queue's service rate through real-time OF directives can lead to a severe performance degradation for the flows traversing that queue because, without a proper rate assignment, it is very difficult to guarantee Quality of Service requirements to the flows [10].

A possible solution to mitigate the performance degradation involves the re-routing of the flows experiencing a violation of deadline constraints (e.g., the flows that are totally received beyond the fixed time constraint) [20] on less congested paths or queues. The underlying idea is that, since we cannot change the service rate of the queues, we act on the ingress traffic, moving a subset of flows on different paths or queues in case of need. In order to be 100% compatible with current OF hardware, we impose no changes to OF specifications and directives. Instead we propose controller modifications and, in particular, we use one popular SDN controller: Beacon [21]. Our updated controller will receive statistics about queues, flows and ports from OF switches and will compute an estimation of the flow rates and of the packet loss of the queues. Based on customizable policies, BeaQoS will be able to select a subset of flows experiencing congestion over the bottleneck queue and to re-route them on another and less congested queue, so improving the switch performances. The action of flow re-routing may be exploited not only for deadline management but also for efficient queue load balancing. On the other hand load balancing is often seen as an action to prevent congestion and, consequentially, to limit and delay performance detriment.

3.5 Related works

Despite traffic engineering (TE) approaches are often ruled by MPLS-TE [22, 23], the ability of the SDN controller to receive (soft) real-time information from SDN devices and to make decisions based on a global view of the network, coupled with the ability of "custom"-grained flow aggregation inside SDN devices, makes TE one of the most interesting use cases for SDN networks.

Global load balancing algorithms are proposed in [24] that addresses load-balancing as an integral component of large cloud services and explores ways to make load-balancing scalable, dynamic, and flexible. Moreover [24] states that load-balancing should be a network primitive, not an add-on, and presents a prototype distributed load-balancer based on this principle.

[25], shows that the controller should exploit switch support for wildcard rules for a more scalable solution that directs large aggregates of client traffic to server replicas. [25] also presents algorithms that compute concise wildcard rules that achieve a target distribution of the traffic and automatically change load-balancing policies without disrupting existing connections. Furthermore, the authors implement these algorithms on top of the NOX OpenFlow controller, evaluate their effectiveness, and propose avenues for further research.

The work presented in [26] shows a system that re-configures the network's data plane to match current traffic demands by centrally controlling the traffic that each service sends on a backbone connecting data-centres. [26] develops a novel technique that leverages a small amount of scratch capacity on links to apply updates in a provably congestion free manner, without making any assumptions about the order and timing of updates at individual switches. Further, to scale to large networks in the face of limited forwarding table capacity, [26] greedily selects a small set of entries that can satisfy current demands and updates this set without disrupting traffic.

Reference [27] analyses a partially deployed SDN network (a mix of SDN and non-SDN devices) and shows how to exploit the centralized controller to get significant improvements in network utilization as well as to reduce packet losses and delays. [27] shows that these improvements are possible even in cases where there is only a partial deployment of SDN capability in a network. The authors formulate the SDN controller's optimization problem for traffic engineering with partial deployment and propose a fast Fully Polynomial Time Approximation Schemes (FPTAS) to solve it.

This last problem is also tackled in [28] that introduces a traffic management method to divide, or to "slice", network resources to match user requirements. [28] presents an alternative to resort to low-level mechanisms such as Virtual LANs, or to interpose complicated hypervisors into the control plane, by introducing an abstraction that supports programming isolated slices of the network. The semantics of slices ensures that the processing of packets on a slice is independent of all other slices. They define their slice abstraction, develop algorithms to compile slices, and illustrate their use by using examples. In addition, [28] describes a prototype implementation and a tool to automatically verify formal isolation properties.

In order to support traffic engineering in the SDN environment, OpenFlow Management and Configuration Protocol (OF-Config) has been proposed. OF-Config [29] is a protocol developed by the Open Networking Foundation

	Queue Configuration	
Performance Metric	1-queue	3-queue
BF - packet loss	25%	71.16%
DF1 - percentage of flows matching the deadline	11.43%	74.29%
DF2 - percentage of flows matching the deadline	17.39%	19.57%

Table 3.3: Performance metrics of the traffic for 1-queue and 3-queues configurations.

used to manage physical and virtual switches in an OpenFlow environment. This tool tries to give to network engineers the ability to set policies and to manage traffic across devices.

3.6 Motivations

Some approaches consider a single queue for each outgoing interface. In order to support QoS mechanisms and traffic differentiation, it is common to configure multiple queues in advance [10]. In order to highlight the importance of traffic differentiation we perform a first group of simulations reported in Table 3.3.

Flow entries mapped to a specific queue will be treated according to that queue's configuration in terms of service rate. Most of the previously mentioned approaches assumes the ability of SDN/OpenFlow to set the service rate of the queues in each SDN device. This chance would be very helpful to improve the SDN switch performance, as would be clear from a second group of simulations reported in Table 3.4.

	Queue Configuration	
Performance Metric	Fixed Rate	Variable Rate
BF - packet loss	0%	0%
DF1 - percentage of flows matching the deadline	100%	100%
DF2 - percentage of flows matching the deadline	25%	100%
DF - percentage of flows matching the deadline	34.78%	100%

Table 3.4: Performance metrics of the traffic for fixed and variable service rate.

Table 3.3 shows the results of simulations we ran aimed at showing how it is hard, without traffic differentiation, to guarantee deadline requirements. During $120s$ of simulation, a SDN Switch $s1$ receives a mix of traffic, generated with *iperf*, composed of "Background flows" (BF) and "Deadline flows" (DF). DF are the flows for which there is an associated deadline: the flow is useful if, and only if, is completely received at the destination within the deadline. On the contrary BF have no associated deadline. BF are CBR flows with a rate randomly chosen in the set $\{50, 60, 70, 80\}\,kbit/s$. DF are divided into two classes: DF1 and DF2. DF1 has a 5 seconds deadline, while DF2 a 9 seconds constraint. The overall traffic descriptors and requirements are defined in Table 3.5.

We tested two configurations by using 125 generated flows. In the first one $s1$ has 1 queue on the outgoing interface (q_0) with a FIFO (First Input, First Output) service rate $s_{q_0} = 3\,Mbit/s$, whereas in the second one it has 3 queues, each of them dedicated to a specific traffic: q_0 for BF, q_1 for DF1 and q_2 for DF2. The service rate of the queues (set in advance) are $s_{q_0} =$

Traffic Class			
Name	Traffic Descriptor	Percentage of Overall Traffic	Deadline Requirements
BF	$50 - 80\,kbit/s$ x $50\,s$	30%	-
DF1	$4.5\,Mbit/s$ x $1\,s$	55%	deadline: $9\,s$
DF2	$1.5\,Mbit/s$ x $1\,s$	15%	deadline: $5\,s$

Table 3.5: Traffic classes and their deadline requirements

$300\,kbit/s$, $s_{q_1} = 1.7\,Mbit/s$, $s_{q_2} = 1\,Mbit/s$. The QoS metrics considered here are the packet loss rate in percentage for BF and the percentage of flows matching the deadline for DFs as shown in Table 3.3.

As one can note the 3-queue configuration consistently improves the percentage of flows matching the deadline and penalizes the packet loss rate of BF. Setting the service rates of simple queues differently, the performances will change but it is clear that traffic differentiation through multi-queues interfaces gives the fundamental gears to manage deadline flows and to tune the level of performances of the network traffic.

Table 3.4 shows the results of the second set of simulations we ran aimed at showing how the power to change the service rate of the queues can improve the deadline management performances. As the previous simulation, $s1$ is receiving a mix of traffic composed of BF, DF1 and DF2. Again two configurations are tested with the same number of generated flows. In the first configuration, $s1$ has 3 queues with a pre-fixed service rate: $s_{q_0} = 2\,Mbit/s$, $s_{q_1} = 4\,Mbit/s$ and $s_{q_2} = 4\,Mbit/s$, whereas in the second one, q_1 can grab the spare capacity from the other two when it needs more bandwidth: s_{q_1} is in the range $[4 - 10]\,Mbit/s$.

The variable rate configuration consistently improves the total percentage of flows that match the deadline, leading it up to 100%, without any impact on BF packet loss. In Table 3.4 the label DF tags the Deadline flows without distinction between DF1 and DF2.

Unfortunately, as highlighted at the beginning of this chapter, current OF specification [1] is not able to configure queues' service rate and delegates this task to an external dedicated configuration protocol: "***Queue configuration takes place outside the OpenFlow protocol, either through a command line tool or through an external dedicated configuration protocol.***" ([1], Section 7.3.5.8). As a consequence, this paper, even if applies multiple queues for traffic differentiation, supposes queue's service rate set and unchangeable in a SDN switch.

3.7 Possible solutions

3.7.1 General Idea

Although the design and implementation of a new OpenFlow directive able to configure the queues' service rate would be the best solution in terms of performances, this choice would come up with a main drawback: it would be totally incompatible with current OF switches that would not take any benefit from the directive.

For this reason we propose an alternative solution *totally compatible* with current OF switches. The underlying idea is shown in Figures 3.1 and 3.2.

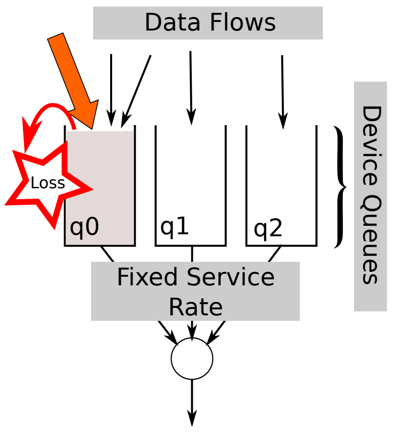


Figure 3.1: Congestion at one of the queue.

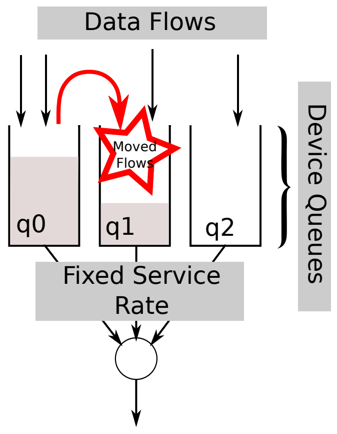


Figure 3.2: Action of re-routing of some flows.

Let us suppose that, during the network operation, the OF switch in Figure 3.1 receives 5 flows that manages through 3 outgoing queues q_0, q_1 and q_2. Let us suppose that the orange flow (i.e. the largest arrow) increases its data rate so that q_0 receives more packets than those it can handle. q_0 incoming rate is higher than the pre-configured service rate. In this situation, increasing the incoming rate eventually leads to packet loss and to a severe reduction of the quality experienced by the flows in q_0. Being unable to change the service rate of the queue, a possible solution involves the re-routing of some flows arriving at q_0 to another queue (e.g., q_1 in Figure 3.2) in order to reduce q_0 incoming rate. Re-routing mechanisms attempt to use the spare bandwidth unused by other queues for reducing the load of more congested queues.

Since we want to keep simple both OF switches and OF specification, we design and implement re-routing mechanisms inside the SDN controller.

Even if the idea is simple, the design of re-routing mechanisms involves functionalities of the SDN controller and, in particular, the following features/ requirements:

- no primitives shall be modified with respect to the current OpenFlow standard.
- the compatibility with early versions of OpenFlow (which is obviously a must);
- the creation of a module able to handle statistics;
- the implementation of the proposed approaches;

The idea of re-routing and the strategies proposed in this paper can be exploited both for specific deadline management purposes, and in the context of the optimal management of hardware resources provided to common software routers. Software routers can run on off-the-shelf general-purpose CPUs and commodity hardware, rather than on expensive dedicated hardware. Commodity hardware not only maintains a high level of programmability and flexibility but is more cost-efficient than specialized hardware solutions and network components. For this reason, software routers are largely widespread [30]. On the other hand, it has been proved that the CPU is the main bottleneck in a software router. Recent advances propose to increase the packet processing performance through parallel processing based on off-the-shelf multi-core processors [31]. In more detail, current software routers implement filters for multi-queue NICs (Network Interface Controllers) used to address incoming packets to a certain queue based on specific packet attributes. By these filters, NICs are able to efficiently distribute the incoming packet processing workload across multiple CPU cores. This also ensures that each packet of a specific flow is served by the same CPU core so avoiding, for example, packet reordering [32].

Instead of using dedicated hardware filters provided by NICs we propose a flexible solution based on the OpenFlow architecture. Our approach consists in using an OF software controller which can monitor incoming flows and has the intelligence to decide the correct queueing strategy. We develop a series of control algorithms able to re-arrange flows in order to make lighter the computational burden of the CPU by equally distributing flows among the available queues.

3.7.2 Implementation: BeaQoS

We chose Beacon [21] as SDN controller. Beacon is a multi-threaded Java-based controller that relies on OSGi and Spring frameworks and it is highly integrated into the Eclipse IDE. Anyway, independently of the specific choice of the controller, our modifications can be implemented in any controller. The structure of the controller consists of a group of functions (called bundles) with dedicated functionalities. The main bundle we focused on is the *Routing* one, which takes care of finding the correct path between the source and destination to forward packets. Moreover, we created an ad-hoc bundle, called *Statistics*, to the purpose of collecting and processing the statistics of the reply messages provided by network switches. The principal proposed modifications of Beacon are:

Statistics Polling Beacon controller has been modified in order to send statistic requests to the switches. We added a function that triggers the dispatch of statistic and feature request messages with a polling interval (PI) configurable through an external properties file. We also designed and implemented a class dedicated to the creation of statistic request messages, such as *ofp_flow_stats_request*, *ofp_port_stats_request*, *ofp_queue_stats_request* [1], in order to obtain useful information about the status of flows, ports and queues respectively.

Statistics This module has two main functions: one is devoted to the creation of the data structures needed to generate a database of statistics related to the network nodes, the other one is dedicated to implement the collection of data extracted from the messages about statistics. The reply messages obtained from the network switches are the introduced *ofp_flow_stats*, *ofp_port_stats*, *ofp_queue_stats*. In addition to the basic statistics that the OpenFlow protocol 1.0 makes available, we added specific functions to the controller, which allow BeaQoS to exploit the collected data in order to compute parameters useful to apply the chosen strategy. The additional statistics computed by BeaQoS, compared with the ones available in OpenFlow 1.0, are shown in Table 3.6.

The main extracted feature is the Estimated Rate (ER) for ports, queues, and flows. We computed the Estimated Rate ER^t at a given time instant as follows:

$$ER^t = \frac{TB^t - TB^{t-1}}{PI} \tag{3.1}$$

in which t is the sampling instant, TB^t are the transmitted bytes at the current instant, TB^{t-1} are the transmitted bytes at the previous sampling instant and PI represents the polling interval in seconds.

Statistics available in OpenFlow 1.0		Statistics computed by BeaQoS
Tx Bytes per Flow	$\rightarrow$	Estimated Rate per Flow
Tx Bytes per Port	$\rightarrow$	Estimated Rate per Port
Tx Bytes per Queue	$\rightarrow$	Estimated Rate per Queue
Flow Match Flow Actions Queue ID	$\}\rightarrow$	Flows per Queue

Table 3.6: BeaQoS Statistics compared with OpenFlow 1.0 statistics.

Obviously the quantity "transmitted bytes" and, consequently, the expression in (3.1), may be applied to ports, queues, and flows. Another parameter we extracted is the number of flows currently belonging to a specific queue (Flows per Queue).

Routing This module has been modified so as to implement the proposed algorithms. When a switch receives a new flow, it contacts the controller in order to know where to forward the traffic. When the controller has to assign each flow to a specific queue, it checks a variable that identifies the algorithm to run. BeaQoS performs a routine to select the correct queue based on the chosen strategy and then notifies the node through the installation of a flow modification.

The proposed approaches are described in detail in the following section.

3.8 Re-Routing Strategies Analysis

In this section, we present two main scenarios in which we compare different proposed re-routing algorithms to find the most efficient solution. The first scenario deals with the problem of the priority flows that must be served within a specific deadline, as introduced in Section 3.6. The second one faces the issue of balancing the load among different queues in a single SDN node.

3.8.1 Deadline Management Scenario

In this scenario we consider both “Background flows” (BF) and “Deadline flows” (DF). As previously described, DF are flows for which there is an associated deadline: the flow is useful if, and only if, it completes within the deadline [20]. DF are of interest in datacenter applications (e.g., web search, social networking) where user requests need to be satisfied within a specified latency target and when the time expires, responses, irrespective of their completeness, are shipped out (Today’s online services have service level agreements (SLAs) baked into their operation [33, 34, 35]). Moreover, online services have a partition-aggregate workflow, being user requests partitioned among (multiple) layers of servers (workers) whose results are then aggregated to form the response. The combination of latency targets and partition-aggregate workflow has implications for the traffic inside the datacenter. Specifically, for any network flow initiated by these workers, there is an associated deadline.

We propose and implement two schemes in order to provide a basic support for deadline management inside a SDN network with mixed traffic BF, DF1 (each flow with *deadline*1) and DF2 (each flow with *deadline*2). To clarify the description of these approaches we assume that all interfaces of each switch are configured with three queues: q_0, q_1, q_2. q_0 is dedicated to BF, whereas the others are used for DF1 and DF2, respectively. The schemes are the following:

Dedicated This scheme assigns each traffic class to a specific queue of the considered switch port. Upon the arrival of a new flow inside the switch, the routing engine of the Beacon controller decides which queue to choose based on the traffic descriptor of the flow. BF are enqueued on q_0, DF1 are assigned to q_1 and DF2 are assigned to q_2.

Deadline This scheme is triggered when the controller receives a request from a switch on how to manage an upcoming flow. The routing module checks the Type of Service field[1]: BF are enqueued on q_0, whereas for DF1 or DF2, the controller chooses the less utilized queue q_{i^*}. The utilization of the queues is computed based on the following function $U(q_i)$:

$$i^* = \arg\min_{i=1,2} U(q_i); \quad U(q_i) = s_{q_i} - \sum_k target_k \cdot n_{k,q_i} \quad i = 1,2 \tag{3.2}$$

being: s_{q_i} the service rate of q_i, known a-priori and configurable from an external properties file; k the index that spans among the classes of service (here DF1 and DF2); $target_k$ the rate we need to guarantee to the flow of class k[2]; n_{k,q_i} the number of flows belonging to the class k

[1] We choose the ToS field to differentiate DF1 and DF2 having in mind the DSCP (Diff Serv Code Point) bits in the ToS field, but other solutions can be implemented.

[2] For example, a flow of class k with size of $100\,kByte$ and a deadline of $10\,s$ needs a $target_k \geq 10\,kByte/s$.

Queue ID	Service Rate	Buffer Size
q_0	$0 - 3\,Mbit/s$	$1000\,packets$
q_1	$2\,Mbit/s$	$1000\,packets$
q_2	$1\,Mbit/s$	$1000\,packets$

Table 3.7: Queue configurations.

and assigned to q_i.

The aim is to maximize the number of DF whose deadline is matched, even at the expense of background flows, if necessary.

We carried out the performance analysis on a PC running Mininet (version 2.1.0) [36]. The scenario is composed of two hosts connected to a SDN switch. The chosen implementation of the switch is Open vSwitch 2.0.2 [37], managed by an instance of BeaQoS running on the same machine. Each port of the switch is configured with 3 queues, q_0, q_1, q_2. The rate assigned to each buffer is shown in Table 3.7. The overall service rate is $3\,Mbit/s$. The queue dedicated to BF has a variable service rate ranging from 0 to $3\,Mbit/s$: this implies that q_0 can be served only if the priority queues are not using the entire link bandwidth. Queue service rates are configured through the Traffic Control (*tc*) module in Linux Kernel.

The traffic used for these simulations, generated through the *iperf* tool, consists, as said above, of 3 types of flows, BF, DF1 and DF2 composed of the following percentages and features: 30% of the overall traffic is BF, which is characterized by a random rate chosen in the set $\{50, 60, 70, 80\}\,kbit/s$ and a duration of $50\,s$; 55% is DF1, generating data at $4.5\,Mbit/s$ for $1\,s$ and,

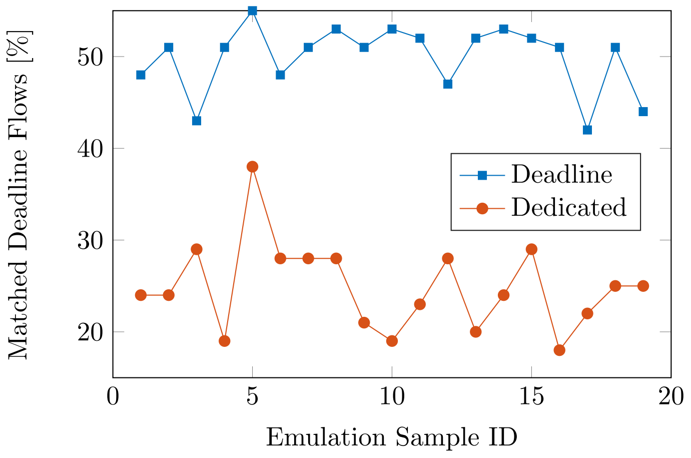


Figure 3.3: Percentage of flows that satisfy the deadline, computed with $H = 100$.

undergoing a deadline of $9\,s$; and 15% is DF2 with $1.5\,Mbit/s$ data rate for $1\,s$ and with a deadline of $5\,s$. The summary of traffic descriptors and requirements are reported in Table 3.5, already used for the results in Section 3.6.

In this scenario we compare the performances of the two different proposed solutions: Dedicated and Deadline.

We ran an emulation of 3 hours of duration composed of 3000 flows structured into BF and DF flows as described above. We present the obtained values averaged over an $Horizon$ (H) of consecutive flows. Each averaged value is called Emulation Sample ID. The metrics used to compare the proposed approaches are the percentage of Matched Deadline Flows (e.g., the percentage of flows satisfying the deadline) and the Loss of Background Flows (i.e. the percentage of lost packets of BF flows). For what concerns Figures 3.3 and 3.4, showing the Matched Deadline Flows, an $Horizon$ $H = 100$ and

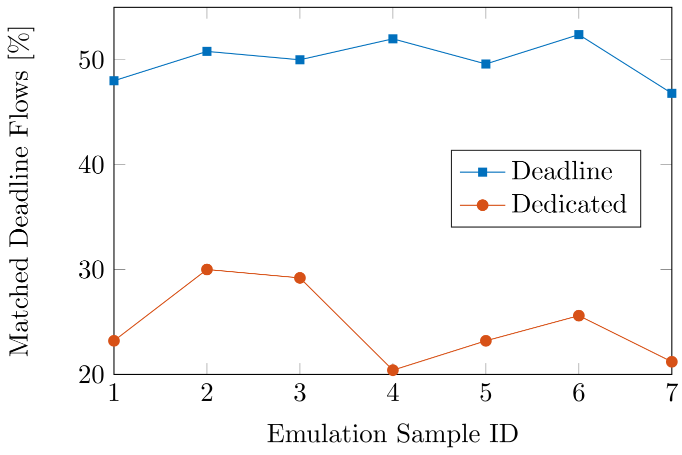


Figure 3.4: Percentage of flows that satisfy the deadline, computed with $H = 250$.

$H = 250$, respectively, is applied taking into account only DF flows. Figures 3.5 and 3.6, showing the Loss of Background Flows, apply again $H = 100$ and $H = 250$, respectively, but involving only BF flows.

Intuitively, large H values capture the steady state of the system and small H values present more measurement noise. Instead of choosing a specific H or trying to capture a flat steady state behaviour, we decided to track the performances over fixed time horizons in order to obtain a more realistic approach, as discussed in [38] and [39]. The results of these tests show that the Deadline scheme allows satisfying the time constraints of a much larger number of DF than the Dedicated scheme. In practice, the Deadline scheme is able to double, on average, the performance of the other approach, referring to Matched Deadline Flows (Figures 3.3, 3.4). The improvement of the number of DF flows matching the deadline is obtained at the expense of BF traffic, which suffers from a much higher packet loss than in the Dedicated

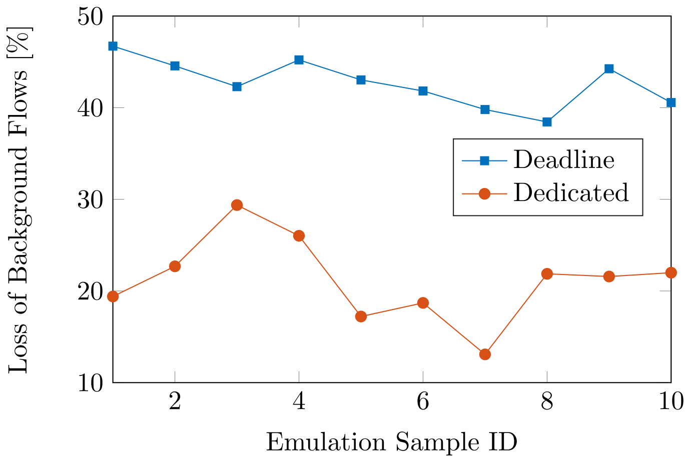


Figure 3.5: Percentage of lost packets for Background Flows, computed with $H = 100$.

scheme, as shown in Figures 3.5 and 3.6.

In short independently of the H value, the Deadline technique is better than the Dedicated one with respect to the percentage of satisfied deadlines for DF flows, at the cost of increasing the loss achieved on BF flows.

Even if the flow specifications are not adherent to a specific real environment, the conducted experiments prove the effectiveness of the presented algorithms. This is true also for what concern the loss of background flow, which, as stated in this chapter, are considered expendable with respect to deadline ones.

3.8.2 Queue Balancing Scenario

As far as load balancing strategies are concerned, we propose three schemes aimed at equalizing the traffic burden in each queue. In order to better illustrate the operating principles of our solutions, we assume a network

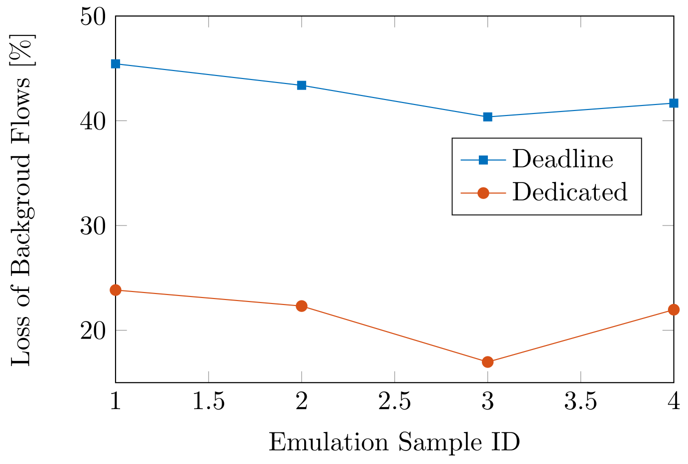


Figure 3.6: Percentage of lost packets for Background Flows, computed with $H = 250$.

scenario in which each interface of each switch has four available queues, q_0, q_1, q_2 and q_3. The service rate of the outgoing interface is equally divided among the different queues. The proposed schemes are the following:

Min Load This scheme consists in assigning the upcoming flow to the least loaded queue. This task is performed by the routing module of the BeaQoS controller. When a new flow reaches a SDN switch the controller checks the estimated rate (computed as in Equation (3.1)) of the queues belonging to the considered output port and selects the one which has the minimum value.

If we think to the rate of the flows as numbers, it is possible to model the load balancing problem among the available queues as a problem of partitioning a given set of numbers into a collection of subsets so that the sums of the numbers in each subset (i.e. the queues of the switches) are as close as possible [40]. This problem is already known in literature as Multi-Way

Number Partitioning and it is NP-complete. For the sake of simplicity we choose to implement an algorithm, which we call Multiway, based on the greedy heuristic described below.

Multiway In this scheme all the flows are queued into q_0 at the beginning, then the controller periodically runs a scheme that sorts the flows in decreasing order based on the computed Estimated Rates (ER) in Equation (3.1) and assigns each flow, analyzed by following the established ER decreasing order, to the queue with the lower utilization so far, in order to equalize the load among the queues.

N-Migrations When the number of flows is huge, the Multiway approach tends to become computationally heavy since it has to analyze and possibly move all the flows traversing the interface. For this reason we introduced the N-Migration strategy, where the number of flow migrations is limited to N. The algorithm runs on scheduled times and iterates N times a routine which selects a flow from the most loaded queue and re-routes it in the least loaded one. The flow selected by the strategy is the one which assures the best load equalization among the queues. This selection is performed evaluating all the possible outcomes through a simple simulation of re-routing.

Although these strategies may seem similar, the performance results are different. Tests about Queue Balancing use a very similar Mininet topology as described for the Deadline scenario. The overall rate availability is $4\,Mbit/s$. The main difference is in the configuration of the queues inside the OpenFlow switch: each interface of the switch has four queues, q_0, q_1, q_2, q_3 and the rate

of the outgoing interface is equally divided among the different queues such as each one has $1\,Mbit/s$ available.
The traffic used in these simulations was generated by using the *iperf* tool and consisted of flows with a rate randomly chosen in the set $50, 60, 70, 80, 90, 100$ $kbit/s$. Flow duration is $50\,s$.
The network was tested with increasing workloads: $100, 125$ and 150 flows running with different seeds.

To better analyse the results, we introduce a performance index that provides a measure of accuracy of our algorithm with respect to the optimal solution, which ideally allows getting the exact amount of traffic in every queue to get load balancing. We call this parameter *index* and we compute it at each time instant t as:

$$index^t = \frac{\sum_i \left(r_{q_i}^t - \overline{r}^t\right)^2}{4}, \qquad t = 0, 1, \ldots \tag{3.3}$$

where $r_{q_i}^t$ is the measured output rate of queue q_i and $\overline{r}^t$ is the optimal queue rate, both evaluated at time instant t. In other words, $index^t$ is a measure of the distance between our solution and the ideal one.

The following plots show the Cumulative Frequency (CF) of $index^t$. CF is defined as the number of occurrences over the total samples in which the $index^t$ is below a certain threshold (*index-th*). Figures 3.7, 3.8 and 3.9 show CF versus *index-th*, for Min Load, Multiway and N-Migrations in case of 100, 125 and 150 flows, respectively. For what concerns the N-Migrations approach, the N parameter is set to 1 for all simulations.

The results highlight that, in all examined cases, Min Load and Multiway schemes show a very satisfying behaviour and have better performances with

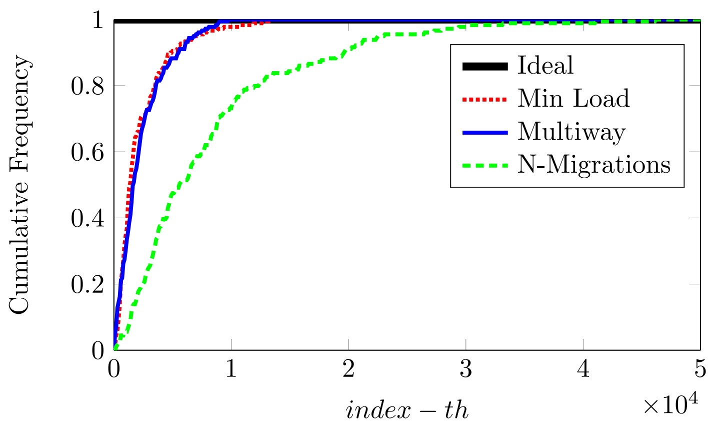


Figure 3.7: Queue balancing performances with 100 flows.

respect to the N-Migrations approach. For example, when we consider 100 flows inside the network, as shown in Figure 3.7, we can say that, in 90% of cases, the distance between our solution and the ideal one doesn't exceed 5000 for what concerns Min Load and Multiway strategies. On the contrary, N-Migrations accuracy curve has a less steep trend than the alternative solutions: the value of *index* for this approach is below 5000 in 50% of cases. In particular it is important to note that Min Load and Multiway behaviours are very close to the Ideal one (CF is 1 for any *index-th* value, including 0) and overlap it for a relatively small *index-th*.

Also the simulations involving 125 and 150 flows confirm the same behaviour, as shown in Figures 3.8 and 3.9.

Concerning N-Migrations: the results show that the N-Migrations approach cannot achieve the same performances of the Min Load and Multiway. This is due to the choice of the N parameter, which is the key of the algorithm. This parameter can be set in order to tune the performances of this approach: as the N parameter grows, the behaviour of the algorithm approaches the Mul-

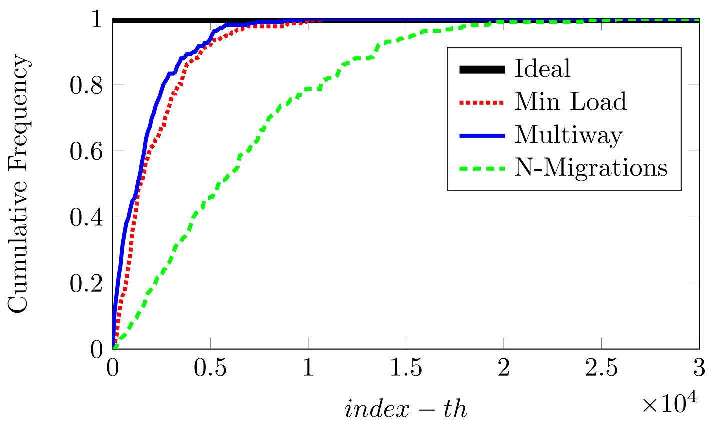


Figure 3.8: Queue balancing performances with 125 flows.

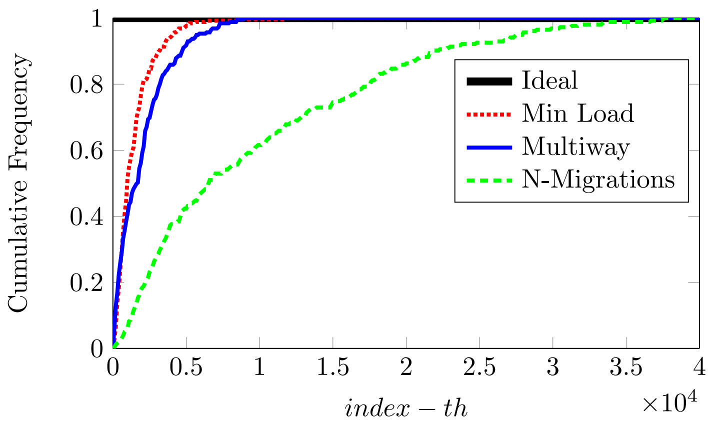


Figure 3.9: Queue balancing performances with 150 flows.

tiway scheme. The choice of the N parameter leads to a trade-off between performance and computational complexity.

3.9 Considerations

3.9.1 Scaling Performances

Concerning statistics (see Table 3.6) acquisition: the types of messages sent by the controller are flow, queue and port requests that are used to gather information about port rates, queue rates and individual flow statistics. The controller receives three statistic replies, one for ports, one for queues and one dedicated to all flows traversing the OpenFlow switch in a given instant.

Since the maximum information sent through the Ethernet frame is 1500 byte, each flow statistics reply message can report only the information about 10 flows. For this reason the number of flow statistic packets in the case of f flows is $\lceil f/10 \rceil$. Given N the number of switches composing the network and considering another two packets for port and queue statistics, the number of packets p that the controller must process at every polling interval is

$$p = \left(\left\lceil \frac{f}{10} \right\rceil + 2 \right) \cdot N \tag{3.4}$$

Considering a significant number of flows f and switches N, the number of packets p received by the controller can be large. This is the price of a fine-grained control of an SDN network at flow-level (*IntServ*). The number of p can be reduced by using the flows statistics for a small number of “aggregate” flows. This could reduce the fine-grained control but relieves the controller from the management of a large number of packets.

3.9.2 Switch Coordination

Even if we show the results by using a single OpenFlow switch in the network, it is possible to extend the concept across multiple SDN devices. The routing module implemented in the BeaQoS controller can manage more then one single switch. For each switch the controller computes all the needed parameters in order to provide the best behaviour, given the chosen algorithm. In order to extend this concept to the entire network, given a specific path to the destination, it would be possible to compute the optimal queue q_{i^*} for each switch belonging to the specific path.

Alternatively, since the controller BeaQoS has the view of the entire network, another possible solution is to examine all existing paths between source and destination for the considered flow. The controller could then compute the best path for the specific flow and finally decide the optimal queue q_{i^*} for all the switches belonging to the selected path.

3.9.3 Timing Performances and Overheads in Queue Balancing Scenario

In queue balancing scenario timing performances are essential to guarantee an "almost" instantaneous load balance among queues in each switch. The main difficulty of this approach is due to the remote nature of the actions of the SDN controller that acts as if the actions were internal switch functionalities. The time elapsing from the load imbalance event at the switch and the new queue balance (*queue balance delay*) can be expressed as the sum of several components, as depicted in Figure 3.10.

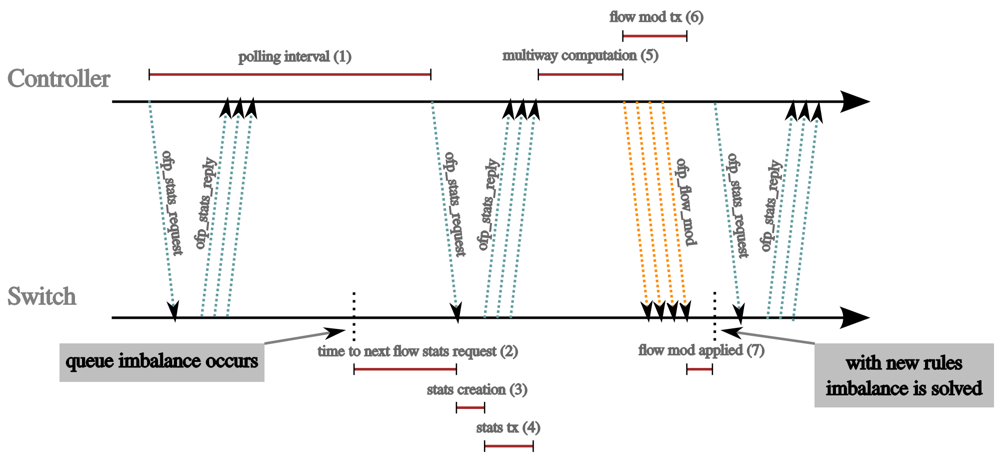


Figure 3.10: Timing Performances in Queue Balancing Scenario.

All our tests are performed with a relatively small number of flows. This allows the controller to manage per flow performances. Considering the Multiway algorithm, the controller can reorder the total amount of flows traversing an SDN switch in a time of the order of milliseconds. Moreover, considering that the controller is connected with the switches using an out of band connection, the time needed to deliver the flow modifications is negligible.

In a large scale scenario with a huge number of flows, it is possible to aggregate flows, reducing the number of sent flow stats and the computation time of the Multiway algorithm.

3.10 Conclusion about Support of Quality of Service in SDN

The impossibility to configure the service rate of the queues in a OpenFlow switch through an OF directive is a limitation that could reduce the quality management capabilities in an SDN network but it is a fact for now. In this chapter, exploiting the re-routing mechanism, we propose a method able to provide a basic deadline management support and an efficient queue balancing without any modification of OpenFlow specifications and switches. We present BeaQoS, an updated version of the Beacon controller able to receive statistics from OpenFlow switches, compute more complex statistics and decide the best queue re-routing strategy. We show the results obtained in performance tests in which we compare alternative Deadline Management approaches and Queue Balancing solutions. Our cases of study show that the proposed solutions allow getting satisfying results when applied to the current OpenFlow environment. Future developments will be devoted to the scalability tests of our solutions and to the study of more complex queue management schemes that could lead to further improvements in performances. We also plan to develop an extension of our internal re-routing approach for the computation of alternative paths between the source and destination, in order to reduce the network congestion.

Chapter 4

SDN in Satellite Environment

The upcoming 5th generation of mobile networks (5G) is specifically conceived to provide extreme flexibility levels by-design to support services and applications with highly heterogeneous requirements in terms of performance, scalability, and deployment scenarios. To cope with these challenging objectives, the current specification of the 5G can be considered as "a network of networks", since it will allow the adoption and combination (as needed by the overlying applications) of different and alternative network stacks and communication technologies. The "virtualization" paradigm is the key cross-cutting enabler of the 5G design. It will pervade the 5G architecture at any layer, in order to provide the related resources "*as-a-Service*".

Clear and tangible examples of this process are Network Functions Virtualization (NFV), Software Defined Networking (SDN), and Software Defined Radio (SDR) technological frameworks, which, together, constitute the "virtualization" engine of the 5G architecture [41]. Such technological frameworks fully decouple hardware infrastructures from network protocols and functions and introduce advanced multi-tenancy capabilities such as the possibility of creating multiple isolated "virtual" domains over the same infrastructure, where multiple tenants can build and run their customized network services. To fully exploit these new capabilities and expose them towards vertical industries and Over-The-Top (OTT) players, the 3rd Generation Partnership Project (3GPP) and Next Generation Mobile Networks (NGMN) Alliance are radically redesigning NorthBound interfaces of telecommunication platforms, by adopting "Network Slicing" [41] as a base service model. The Business/Operational Support Systems (BSS/OSS) of upcoming 5G network platforms are meant to expose "customized" and isolated virtual projections of the mobile network (i.e. Network Slices) to vertical industries and OTT players, so as to enable them to run their applications and services on top of these network slices. To this end, a network slice is composed of a number of logical sub-networks that can have different roles and configurations. Such subnetworks can be instantiated as "private" network projections inside the slice, or shared among multiple slices (e.g., to attach multiple slices to the same radio access network).

The potential role of satellite networking in such ecosystem becomes manifest if referred to this slicing model, within which satellite resources can be embedded, either as Physical Network Functions (PNFs), when considered in their current deployment, or, with much greater relevance, by including their virtualized operational components as manageable entities in the 5G architectural framework. Thanks to their intrinsic ubiquity and broadcasting capabilities, satellite networks can play multiple roles in 5G. The satellite can act as a main single backhaul segment for rural areas, aircrafts, vessels, trains, or as additional backhaul means to opportunistically provide additional connectivity/bandwidth resources, also improving service continuity, or as a pure transport subnetwork.

The integration and use of satellite technology within the 5G ecosystem obviously poses new architectural and service requirements/limitations. For instance, on one side, it is reasonable to assume that satellite subnetworks can be directly applied to those traffic flows (e.g., mission critical data) that are associated with 3GPP 5G [42] Quality of Service (QoS) Indicators (5QI) allowing delays in the order of 1-2 hundred milliseconds. On the other side, satellite subnetworks can be adopted to facilitate and make more effective the deployment and operations of other intermediate 5G subsystems such as edge computing nodes needed to cope with tighter and more challenging 5QI levels, as for Augmented Reality applications. In the edge computing scenario, satellite interconnectivity may be exploited for the unicast/multicast/ broadcast geographical distribution of video, audio, and application software binaries to a large number of terminals simultaneously.

In order to enable this deep integration between satellite and 5G, a number of actions should be undertaken to bring state-of-the-art satellite technologies closer to the virtualization paradigm used within the 5G architecture. Many issues are related to physical layer aspects; quoting [43]: "non-orthogonal multiple access (NOMA), massive multiple input and multiple output (MIMO), cooperative communications and network coding, full duplex (FD), device-to-device (D2D) communications, millimeter wave communications, automated network organization, cognitive radio (CR)". Nevertheless, from the networking viewpoint, virtualization and multi-tenancy are key aspects. Despite satellite technologies are well known to provide advanced network virtualization means, since they allow the dynamic management of multi-point QoS-guaranteed links, these capabilities should be exposed "*as-a-Service*" to multiple concurring tenants. In this respect, the potential impact of architectural frameworks based on NFV, SDN and SDR might be more than relevant.

4.1 State of the Art for SDN/NFV Enabled Satellite Networks

The physical and hardware separation between control and data forwarding nodes is one of the main principles behind the SDN paradigm. Its implementation is based on three different functional planes: Management Plane, whose purpose is to compute resource allocation strategies to provide each user with the required QoS, depending on the user's policies and current status of the network; Control Plane, aimed at computing and enforcing forwarding rules to a number of data forwarding nodes in order to properly route traffic flows; Data Plane, composed of the nodes of the underlying network infrastructure, whose only purpose is to forward the incoming traffic flows, by following the given rules.

The aim of NFV is to decouple network functions from dedicated physical devices, making possible to run such functions on general-purpose servers which could be deployed in network operators' datacenters. In this way, a more precise hardware resource allocation and sharing can be achieved, implementing Virtual Network Functions (VNFs) on virtual machines and assembling and chaining VNFs to create services.

These new concepts can also be employed in satellite communication networks, allowing:

- intelligent delivery and deployment of new services in a flexible and programmable way;
- decrease in energy consumption, by virtualizing the functions performed by the ground segment of the satellite infrastructure and consolidating/activating/deactivating them on remote datacenters;
- Capital Expenditure (CAPEX) decrease by exploiting general-purpose hardware components to deploy virtualized functions;
- Last but not least, the flexible embedding of satellite networking functionalities in the creation and dynamic adaptation of network slices, along with the required resource provisioning at the level of the Satellite Network Operator (SNO).

SDN and virtualization for broadband satellite networks are investigated in [44]. This has been one of the first studies to include a vision of how SDN and NFV concepts could be employed in satellite networks. The authors propose a network architecture based on GEO satellite communications. Reconfigurable broadband satellite networks are also the focus of the research work in [45], where a strategy is developed to deal with the problem of resource management based on a functional architecture composed of virtualized functions distributed throughout the network. [46] proposes a joint placement of controllers and gateways in an SDN-Enabled 5G-Satellite Integrated Network.

An SDN/NFV-based framework for integrated satellite-terrestrial communication networks called SERvICE is considered in [47], which exploits the centralized control of SDN to suggest a strategy to distribute the three planes of the SDN paradigm in the various network nodes of a multi-layer satellite network. The Management plane acts as the orchestrator of the overall network in the Satellite Network Management Center (SNMC). The Control Plane is divided into two parts: the space part, dealt with by the space controller in GEO satellites, and the terrestrial part, in charge of the terrestrial controllers implemented inside datacenters and Satellite Gateways (SGWs). The Data Plane is also divided into space and terrestrial parts and is composed of MEO and LEO satellites, SGWs, and other intermediate terrestrial nodes, such as SDN switches.

In recent years, the growing interest in the next generation of networks has led to an interest in and the proliferation of different project opportunities financed by the bodies of the European Commission and the European Space Agency. These projects are very important for the innovative aspects and the problems associated with the use of new generation networks. Below are some of the main projects concerning 5G networks.

The European H2020 SANSA (Shared Access Terrestrial-Satellite Backhaul Network Enabled by Smart Antennas) [48] project has the objective of increasing the performance of mobile backhaul networks, in order to meet the 5G requirements. Specific goals are to increase the capacity of the backhaul network trying to meet the predicted traffic demand of 5G, to improve the network resilience against link failure and congestion, along with the spectrum efficiency in the Ka band, to reduce the energy consumption of the current mobile networks and to ease their deployment. To these purposes, the project proposes the use of smart antennas to set up a novel end-to-end system architecture composed by both terrestrial and satellite nodes. Flexibility in the network is achieved through a Hybrid Network Manager (HNM), which includes configuration, event and topology management functionalities.

The European H2020 project VITAL (VIrtualized hybrid satellite - TerrestriAl systems for resilient and fLexible future networks) brings NFV into the satellite domain and enables SDN-based resource management in hybrid terrestrial-SatCom networks. A framework named Satellite Cloud Radio Access Network (SatCloudRAN) [49] is defined. Its main principle is to virtualize a DVB (Digital Video Broadcasting) - Satellite Second Generation (DVB-S2)/ DVB - Return Channel Satellite Second Generation (DVB-RCS2) ground infrastructure onto a centralized cloud-based processing platform. Three different virtualization levels are identified: network layer functions, MAC layers functions, and physical layer ones up to the radio frequency front-end of SGW OutDoor Units (ODUs). In detail, in the first level network functions such as Performance Enhancing Proxy (PEP), admission control strategies and QoS policies' management are performed in a centralized hub. IP packets are sent to the SGW. In the second level, the uncoded DVB-S2 frame (called BBFRAME) is created remotely and then sent to the physical gateway. In the last level, data packets forwarded to the ODUs are physical layer frames (I/Q symbols). This framework could allow a full virtualization of the satellite delivery chain and its provision "*as-a-Service*" to multiple tenants contributing to the Satellite Network-as-a-Service (SatNaaS) paradigm

[50].

ARTES 1 CLOUDSAT aims to determine the applicability of SDN and NFV technologies in order to define and validate integrated virtualized satellite - terrestrial architectures [51]. The network architecture is composed of the following subsystems:

- Infrastructure, including the virtualization-capable equipment on which network services are deployed: switches and routers of the satellite terminals, and gateways.

- Infrastructure management entities, based on distributed management paradigms, such as Virtualized Infrastructure Management (VIM) entities for the SDN/NFV enabled segments and the satellite segment, and a Wide Area Management (WAN) entity.

- Orchestrators, in charge of the deployment of services and resource allocation within each network segment.

- Federated Manager, representing the interface toward each orchestrator, as well as the interface toward final users.

4.2 Open Challenges

Despite the research efforts performed to fill the gap between the current satellite communication networks and their envisioned network virtualization evolution, we have identified some open challenges, which require being further investigated and solved before proposing a stable and standardized network architecture. All these issues have a strong impact on the future integration of satellite technologies into the 5G ecosystem; for instance, on how a satellite network may be included in a slice subnetwork, and how it may support dynamic lifecycle operations such as instantiation, de-instantiation, and tuning, as discussed in the next section.

The first issue to be tackled is how to distribute the different layer functionalities that compose the SDN architecture, i.e. in which nodes to locate the three SDN planes. This problem involves different factors, such as the high propagation delays of satellite links and the processing power capabilities of the considered components. Satellite networks may use different types of satellites acting at different altitudes (GEO, MEO, LEO) and characterized by different sizes, such as pico, nano, micro, etc. For these reasons, their communication capabilities are differentiated, in terms of transmission frequency bands, transmission rate, and number of on-board antennas that can be installed. All these variables can lead to different choices about SDN planes positioning, and, consequently, to different satellite network architectures.

Another concern in the design of an SDN satellite network is the implementation of the communication protocol between Data and Control Planes. In traditional SDN networks, this protocol is identified in the de-facto standard OpenFlow. It enables the collection and processing of the network status information in order to allow Control and Management Planes enforcing policies and forwarding rules about current traffic flows. In a satellite network there is the need to collect network status information that may be insignificant in terrestrial networks, such as network topology changes due to satellite movements, satellite current available energy and storage space. To allow this, some extensions of the OpenFlow protocol may be required.

As already mentioned, the network topology may change during the network lifetime, owing to LEO and MEO satellites motion. As a consequence, there is the need of a handover procedure to keep the flow tables of the Data Plane nodes updated, performing new rule computations when needed. Another situation in which handover is required is when a satellite terminal, served by a given satellite, loses its visibility and has to switch to another one [52]. Even in this case, a change of the flow rules inside the involved switches and, possibly, reconfiguration of satellite NFV services may be needed, in order to avoid service interruption. Checking the impact of satellite mobility on virtualization and on the creation of logical virtual networks *as-a-Service* dedicated to given use cases (slices) is indeed a challenging task.

Another open challenge is related to the problem of the gateway diversity. The ground infrastructure may be composed of a set of satellite gateways linked together through the terrestrial network. Therefore, they offer different points of access to the space segment, which are geographically distributed in a wide area. This network topology, if really exploited, implies the application of strategies to choose the best satellite gateway for the forward links [53, 54]. The spectrum frequency bands used by satellite transponders are high, which increases the achievable transmission rates but also the attenuation due to atmospheric phenomena, such as rain. This means that the access to the space segment may be, in a given period of time, more convenient from one point with respect to another, both from the performance and from the energy viewpoint. Selecting the gateway may give practical advantages, if properly orchestrated. A real-time change of the satellite gateway for the ongoing transmissions due to the extreme attenuation of the forward link of the currently selected satellite gateway is a possibility; however, on one side, it should be transparent for OTT players using slices, and, on the other side, it should be dynamically managed by the network control plane in an agile and flexible fashion. For example, slice internal elements (i.e. slice subnetworks) might be reconfigured to route traffic towards the new gateways.

Other open issues regard real-time monitoring and resource constraints, which are not limited to the widely investigated GEO and LEO scenarios. Since the past few years, new kinds of satellites, such as CubeSats, have been attracting the attention of a large number of industries and universities, thanks to their lower costs and shorter deployment. The size and weight of these satellites are much lower if compared to GEO and LEO, but they suffer from very strict constraints about, for example, available energy, storage capacity, and computational power. These variables, among others regarding the status of the satellites in contact with the satellite gateways, should be monitored and controlled in the resource allocation process. At the same time, they make the provision of slices more time-dependent. To cope with the dynamic satellite features, slice provision and adaptation should be performed along with real-time monitoring of performance parameters and resource availability.

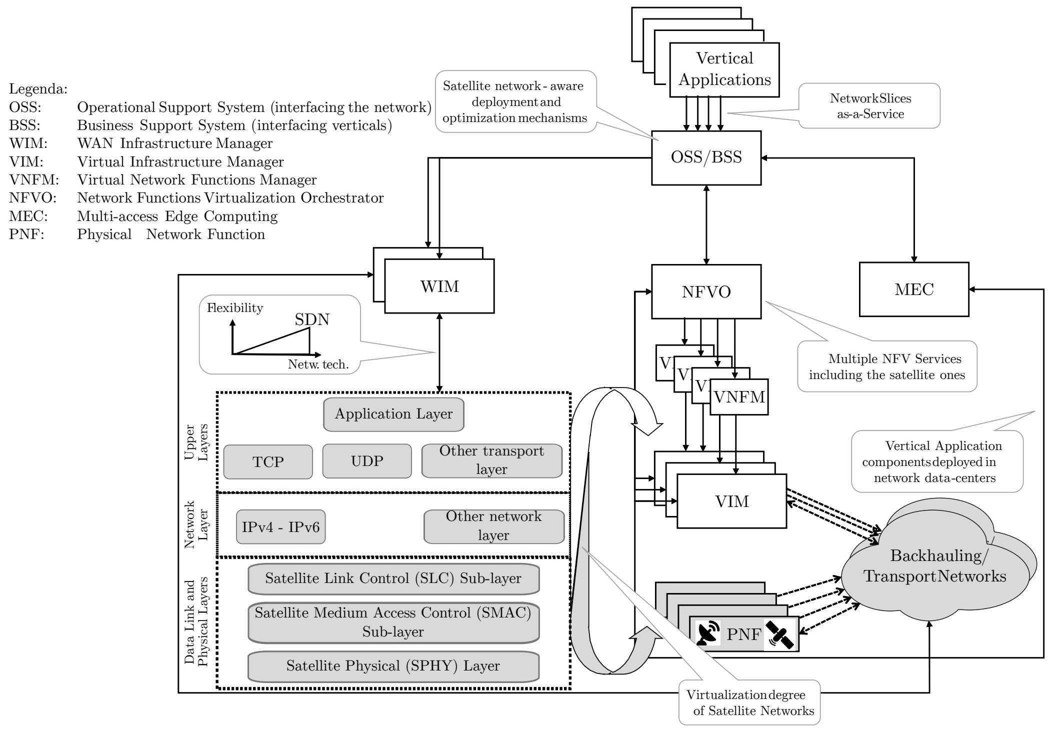

Legenda:
OSS: Operational Support System (interfacing the network)
BSS: Business Support System (interfacing verticals)
WIM: WAN Infrastructure Manager
VIM: Virtual Infrastructure Manager
VNFM: Virtual Network Functions Manager
NFVO: Network Functions Virtualization Orchestrator
MEC: Multi-access Edge Computing
PNF: Physical Network Function
Vertical Applications
Satellite network-aware deployment and optimization mechanisms
NetworkSlices as-a-Service
OSS/BSS
WIM
NFVO
MEC
Flexibility
SDN
Netw. tech.
V
V
V
VNFM
Multiple NFV Services including the satellite ones
Vertical Application components deployed in network data-centers
Upper Layers
Application Layer
TCP
UDP
Other transport layer
Network Layer
IPv4 - IPv6
Other network layer
VIM
Backhauling/ TransportNetworks
Data Link and Physical Layers
Satellite Link Control (SLC) Sub-layer
Satellite Medium Access Control (SMAC) Sub-layer
Satellite Physical (SPHY) Layer
PNF
Virtualization degree of Satellite Networks

Figure 4.1: Architectural Framework

4.3 Proposed Solutions

With reference to the 3GPP, ETSI NFV Management and Orchestration (MANO, `http://www.etsi.org/technologies-clusters/technologies/nfv/open-source-mano`) and ETSI Multi-access Edge Computing (MEC, `https://bit.ly/2IvXVaY`) architectural frameworks, we can refer to the architectural elements depicted in Figure 4.1 to highlight the main points connected with the deployment of satellite-related functionalities and their embedding as full-fledged slice components. Current satellite networking elements can be seen as PNFs, providing long-haul connectivity. To be integrated and orchestrated as slice components by an NFV-Orchestrator

(NFVO), upon requests coming from the OSS to satisfy the requirements of vertical applications, the functionalities of SGWs and Satellite Terminals (STs) need to be virtualized except for ODU, which remains a PNF, basically conforming to the SatCloudRAN paradigm. To better highlight such functionalities and their mapping to VNFs, in Figure 4.1, we have included the representation of a satellite network protocol stack that can implement either standard protocols such as TCP/UDP and IP or dedicated protocols indicated as "Other transport/network solution", with the intention to include proprietary architectural elements aimed at performance optimization such as PEP and header compression. With the desired flexibility, satellite components (physically and/or virtually implemented in VIMs) can then be employed by the WIM in the backhaul, whenever needed to support applications whose KPIs are compatible with their characteristics, or even to create transport links or subnetworks toward the Enhanced Packet Core (EPC). The role of SDN here becomes instrumental to allow fast reconfiguration and interconnection of attachment points for the functional components. In the MEC framework, in the presence of otherwise isolated terminals, the satellite virtual network may be the only means to deploy application components close to their users and to provide them with caching at the edge, in order to satisfy stringent application requirements.

Let us make a practical example. A vertical service request may be monitoring and controlling remote installations such as oil and gas pipelines through SCADA (Supervisory Control and Data) or, alternatively, tracking assets like containers. Remote installations, as well as containers when on board vessels, may be networked only through satellites, but Vertical Applications may ignore this technical need and deliver the service request to the BSS. The OSS checks multiple NFV services exposed by the NFVO and selects the satellite transport providing a given quality of service in terms of delay, loss, and jitter (if requested). To provide the assured quality the satellite network may need to operate specific actions, from the transport layer (e.g., PEP, TCP optimization) and network layer (e.g., IP DiffServ/ IntServ, IP routing), within the Satellite Independent layers, down to link and medium access control and physical layer (e.g., MAC using SIC - Successive Interference Cancellation -, adaptive coding and modulation, etc.) in the Satellite Dependent part. These operations may be performed in a VIM by one or more datacenters, not necessarily located nearby the satellite Earth station, connected to each other by the WIM.

Open challenges identified in the previous section may be mapped over the architectural elements in Figure 4.1, as also shown in Table 4.1.

Challenges	**Involved Architectural Elements**
SDN Planes Positioning	WIM/PNF
SDN Communication Protocol issues	WIM/PNF/VNFM
Gateway Selection	OSS
Real-time Monitoring	OSS/NFVO/PNF
Impact of Satellite Motion on Virtualization	SS/NFVO/WIM
Resource and Performance Constraints issues	OSS/NFVO/VIM/WIM/MEC

Table 4.1: Matching between challenges and Architectural Elements

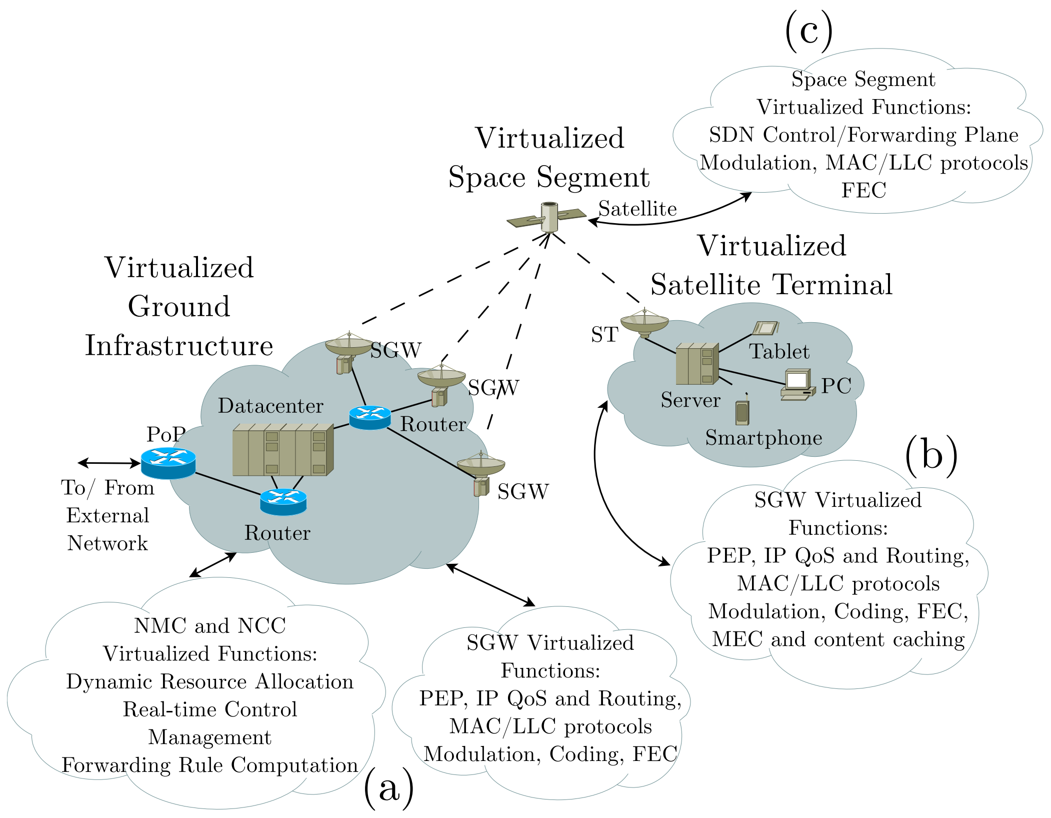


Figure 4.2: Road-map for an SDN/NFV-enabled satellite network

The integration of terrestrial and satellite networks in 5G through the virtualization of network functions, the provision of slices, and the use of general-purpose instead of ad-hoc hardware, will not be immediate. Moreover, the investments required to design and deploy a GEO/LEO satellite communication network are huge, so current satellite operators cannot replace costly hardware components before the end of the scheduled network lifetime, especially concerning on-board technologies.

Before implementing a complete operative case as the one used in the previous practical example, a gradual virtualization would be recommendable to facilitate a preliminary integration in the near future. We have identified three possible incremental virtualization levels, as shown in the clouds (a), (b), and (c) of Figure 4.2, respectively:

(a) *Ground Infrastructure*, physically composed of SGWs (i.e., the nodes interfacing satellite portions and ground infrastructure, which include ODUs), Network Control Center (NCC) and Network Management Center (NMC). The first step could be to virtualize network control and management functions previously performed inside the NCC and NMC, which would be virtually implemented inside a datacenter rather than on ad-hoc nodes. These functions include dynamic network resource allocation, real-time control and non-real-time management of the overall network and could include the actions related to SDN Management and Control Planes, such as user policies management and forwarding rules computation. The functions performed by SGWs can be virtualized and remotely located in one or more datacenters, reducing the specific-purpose hardware components of the SGWs, which could be limited to the ODUs, excluded from the virtualization. As described in [49], there may be three different variants for the virtualization of a SGW, depending on the virtualization "depth": only network and upper layers functions, such as PEP and VPN (Virtual Private Network); network and upper layers + Encapsulation MAC functions; network and upper layers + Encapsulation MAC + Physical layer functions, such as adaptive Forward Error Correction (FEC) coding and modulation, giving access to satellite links.

(b) *Satellite Terminals*. The second step could be to virtualize the functions performed by the STs. Considering their role, the virtualized functions could be the same as for the SGWs except for the scheduling task that the SGW has to perform across many STs that are sharing the same resources. In this case the SGW has to coordinate different STs with different demands, QoS profiles and channel conditions, whereas the STs do not have to deal with this task. Moreover, additional functionalities related to the MEC and content caching paradigms can be implemented inside remote servers to help reduce the latency.

(c) *Satellites*. The final step could involve the addition of virtualized functions on board satellites. Considering the different kinds of satellites and the various possible satellite constellations, both SDN Control and Data Planes functions could be implemented on-board satellites. Satellite communication functions could be virtualized in order to better exploit limited available resources. This point, however, requires a careful analysis of the on-board available resources, both in terms of performance and energy consumption and implementation costs.

4.4 The role of SDN in the 5G Satellite Communications

In this section, we focus on SDN in order to better understand which are the consequences of its employment in satellite networks. The main principle of SDN is to centralize the intelligence of the network decoupling control and Data Planes. SDN can be employed for different purposes. The most common is routing, but also congestion control, flow control, and even security, allowing novel control and management strategies based on fine-grained traffic flow identification.

The most commonly used communication protocol between SDN switches and controllers is OpenFlow [1]. It is used to manage the flow tables inside the SDN switches to forward packets. These tables consist of lists of flow entries which include matching rules and corresponding actions. Each matching rule contains a set of matching fields, i.e. a set of parameters to identify a traffic flow, such as source and destination Ethernet addresses, source and destination IP addresses, source and destination TCP/UDP ports, among others. An action is defined for each matching rule. If there is a match between received packets and matching rules, the packets are forwarded by using the corresponding actions. If there is no match, SDN switches ask for a new forwarding rule which is computed and sent back by the SDN controllers.

Several studies have already investigated the employment of SDN in integrated satellite-terrestrial networks. The main difference among them is how the different authors have decided to locate the different SDN Planes and their functionalities within the satellite network components. [55] investigates the use of the SDN paradigm in High-Throughput Satellite (HTS) networks, identifying the most interesting use cases and perspectives: the used network infrastructure is composed of a Geostationary (GEO) satellite including the SDN controller, Satellite Gateways acting as SDN switches, and a terrestrial Network Management Centre (NMC) implementing the Management Plane. Almost the same network architecture is considered in [56], where an SDN-based Information-Centric Networking (ICN) architecture for an integrated satellite-terrestrial network is proposed. [55] also suggests that a network composed of a Low Earth Orbit (LEO) satellite constellation in addition to a GEO satellite system could bring benefits. A novel software-defined satellite network architecture, called OpenSAN, is proposed in [57]: the satellite ground segment infrastructure acts as the Data Plane, a GEO satellite includes the Control Plane, while the Management functionalities are performed by a terrestrial Network Operation and Control Centre (NOCC). An architecture which exploits LEO inter-satellite links (ISLs) for data forwarding and GEO broadcasting capabilities for rapid network deployment is described in [58]. Other studies have considered a hierarchical multi-layer satellite network where the Data Plane is included both in LEO and Medium Earth Orbit (MEO) satellites [47, 59, 60].

Only [58] considers the computation of the time required to establish a new forwarding rule and to route and deliver the first packet of the new data. The rest of the packets will follow the same forwarding rule. The required time is in the order of few milliseconds in terrestrial networks and it does not significantly affect the application data delivery time. In integrated satellite-terrestrial networks, it could be not negligible, especially considering satellite link delays together with the strict performance requirements of some applications. In this case, the aim of our work is to estimate the mean value of this time in a software-defined integrated satellite-terrestrial network, in order to identify which are the maximum performance requirements a certain application can get if its data are forwarded through satellites.

4.5 SDN-based Satellite Terrestrial Network

In order to understand better the possible implication in the use of SDN in a satellite-terrestrial network, we introduced an example network shown in Figure 4.3, which is composed of three portions. The network is very challenging, in particular for the delay, because it involves GEO satellites, which have a main role also concerning the SDN control architecture, as explained below.

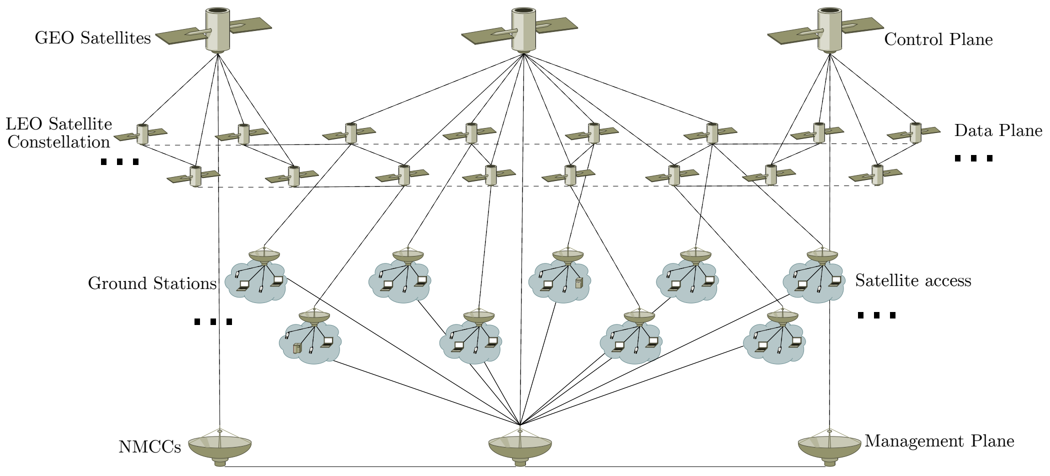


Figure 4.3: Considered terrestrial-satellite network with focus on one of the three equal portions

Each portion is composed of a set of heterogeneous terrestrial networks which could be based on different technologies, such as 5G cells, Local Area Networks, LTE cells, linked together by a terrestrial infrastructure but also by a LEO satellite constellation, which is common among the 3 portions. In this way, traffic flows with given performance requirements, especially the delay-tolerant ones, can be forwarded through the satellite constellation, partially offloading the terrestrial infrastructure. Alternatively, the satellite route could be selected in case of fault of the terrestrial infrastructure.

In details, the overall terrestrial portion is composed of:

- Ground stations: terrestrial stations linked, on the one hand, with terrestrial terminals through terrestrial communication means (wired or wireless) and, on the other hand, equipped with a satellite antenna to send/receive data to/from satellites.
- Terrestrial hosts: they are all kind of nodes which can generate data traffic, such as user mobile and fixed terminals and servers.
- Network Management and Control Centres (NMCCs): three terrestrial centres which manage network resources and user requirements.

The satellite portion includes three GEO satellites. Each GEO satellite covers a third of the shared LEO constellation. LEO satellites are linked with the ground stations and, through ISLs, among them. The considered LEO satellite constellation is a multi-orbit polar constellation where orbital planes are circular and equally spaced among them. Satellites are equally spaced within each orbit. Its representation is reported in Figure 4.4.

There are two types of ISLs: intra-orbit ISLs (ia-ISLs), which are the links between adjacent LEO satellites belonging to the same orbit; and inter-orbit ISLs (ie-ISLs), which are the links between adjacent LEO satellites belonging to different and adjacent orbits. In this way, each satellite has four bidirectional ISLs. However, in this kind of constellation, satellites which are travelling at high latitude are not able to keep ISLs active due to their speed and consequent Doppler effect. Another issue which could affect ISL status

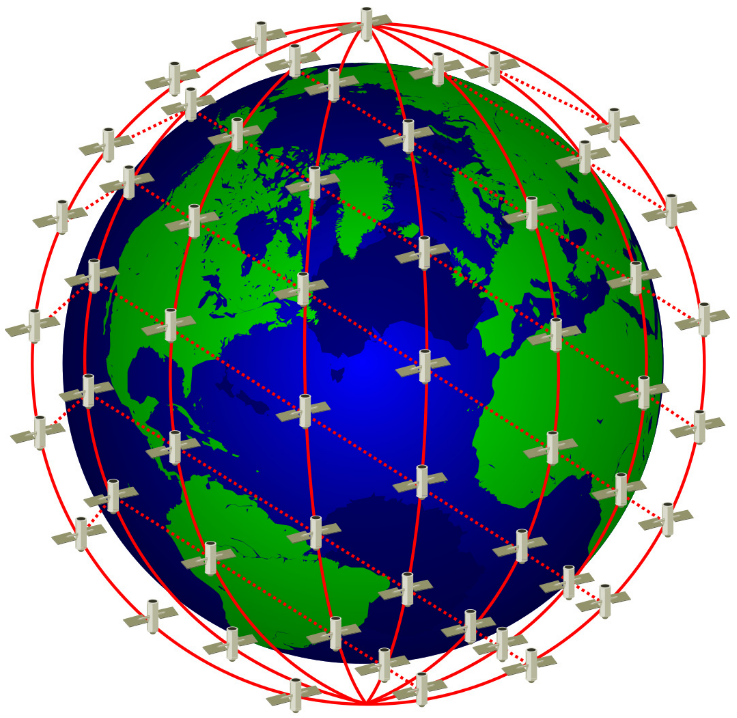


Figure 4.4: Polar LEO satellite constellation

is the presence of the so-called "Seam". Orbital planes are equally spaced around a 180° angle, hence there are two adjacent orbits whose satellites travel in opposite direction (as shown in Figure 22.1 of [61]): the first ones from south to north and the other ones from north to south. Communications through the ie-ISL of these satellites could not be possible for the same reason of the high latitude ISLs. In our network all ISLs are always active except for the ones operating at high latitudes. Besides, we assume a "W" ISL pattern (Figure 22.3(a) of [61]). Figure 4.5 illustrates our considered ISL model: the horizontal size is two times the number of orbits and the vertical size is half the number of satellites per orbit.

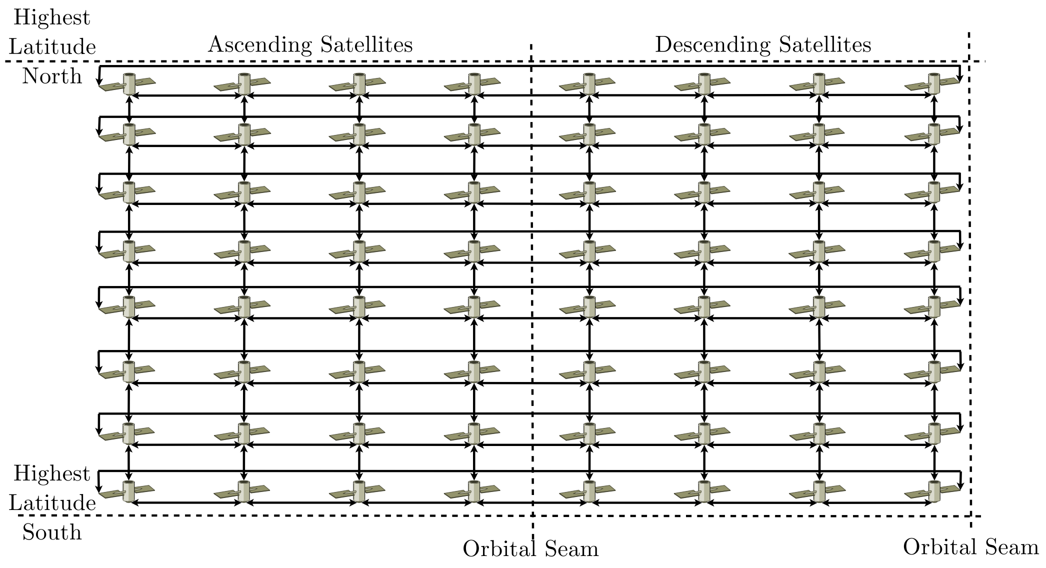


Figure 4.5: LEO satellite constellation ISL model

We structure the Data/Forwarding, Control, and Management Planes as depicted in Figure 4.6.

LEO satellites constitute the Data Plane and act as SDN switches. Their only purpose is to forward the data received from ground stations through the constellation following the routing instructions stored in their memory and previously received by the GEO satellites. GEO satellites act as SDN controllers. Their aim is twofold: each of them periodically collects and sends information about the statistics of the underlying LEO satellite constellation to the related NMCC; every time a GEO receives a new flow entry request, it computes the forwarding rules exploiting the data management strategies received from the NMCC and then sends a flow entry reply to the proper LEO satellite. NMCCs collect and keep updated the information about the

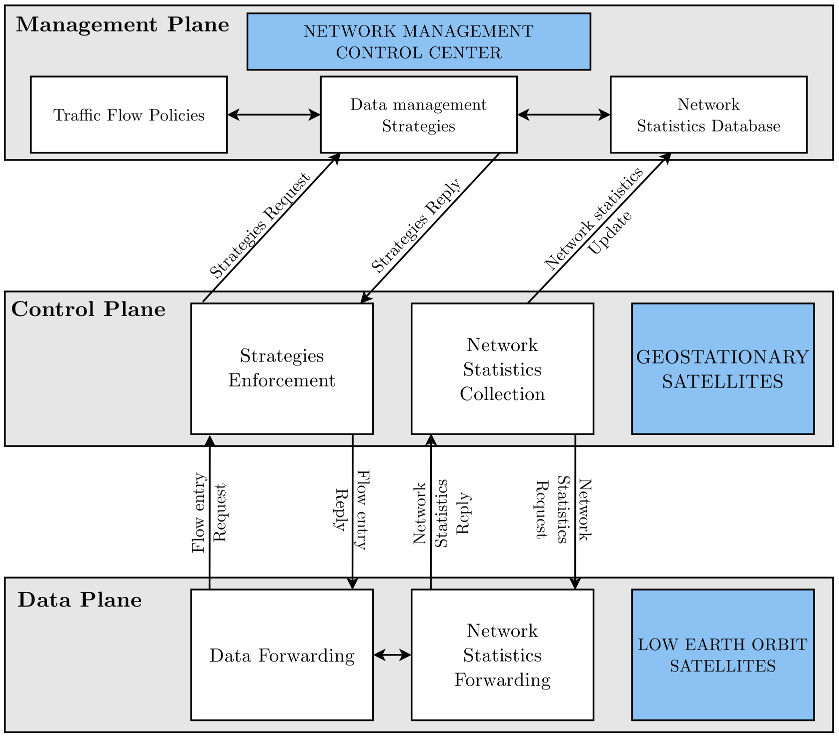

Management Plane
NETWORK MANAGEMENT CONTROL CENTER
Traffic Flow Policies
Data management Strategies
Network Statistics Database
Strategies Request
Strategies Reply
Network statistics Update
Control Plane
Strategies Enforcement
Network Statistics Collection
GEOSTATIONARY SATELLITES
Flow entry Request
Flow entry Reply
Network Statistics Reply
Network Statistics Request
Data Plane
Data Forwarding
Network Statistics Forwarding
LOW EARTH ORBIT SATELLITES

Figure 4.6: SDN Planes and functionalities scheme

network statistics received from the GEO satellites and the traffic flow performance requirements (policies) received from the terrestrial hosts. When NMCCs receive a strategy request from the GEO satellites, they establish a data management strategy exploiting these information and send it back to the GEO satellites. We opt for a distributed Management Plane in order to reduce the access time between NMCCs and GEO satellites, even though, in this case, a protocol to keep the consistency of this information is required, as also mentioned in [57]. In some cases, Control and Management Planes can be implemented in the same node. We have decided to apply the Management Plane on terrestrial nodes in order to reduce the time to install and update traffic flow policies and to avoid storing a huge amount of information and tackling a high computational effort on-board GEO satellites that have storage, energy, and computational power constraints.

Taking into account all these steps, the time required to route a new traffic flow could be quite large, especially considering the delay of GEO satellite links. In the next section, we will estimate the mean value of this time, in order to quantify its impact on the expected performances and identify the maximum performance requirements that allow the use of the satellites.

4.6 Time estimation model

We consider a terrestrial host TH_S generating data packets destined to another terrestrial host TH_D located in areas covered by two different ground stations GS_S and GS_D, respectively. TH_S sends its packets to GS_S which uploads them to the LEO satellite LE_S (which is an SDN switch) in contact in that moment. TH_S's packets are a new traffic flow for LE_S that requires a specific forwarding rule to the GEO satellite GE_S (an SDN controller) which is covering the LEO constellation portion where LE_S is located. GE_S contacts the related NMCC NM_S asking for information about the current status of the network and the performance policy related to the identified traffic flow. A routing path from LE_S to the LEO satellite LE_D (also an SDN switch) in contact with GS_D is computed by GE_S. Proper forwarding rules are sent to each involved LEO satellite/SDN switch. If LE_D is located in an area covered by another GEO satellite/SDN controller GE_D, a portion of the routing path includes the LEO satellites inside GE_D's coverage area. In this case, the required information will be sent also to GE_D through its related NMCC NM_D. When LE_S receives its forwarding rule, the packets are sent through the SDN-based LEO network until they reach LE_D. Finally, LE_D downloads them to GS_D and then to their final destination TH_D.

To ease the model always keeping the considered scenario as close as possible to a real one, we consider only propagation delays. We ignore transmission, queuing, and forwarding delays in this case, because they may be considered negligible compared to the satellite link propagation delays for traffic flows characterized by short duration. We assume that there is no handoff during the overall described process, i.e. the topology does not change. Consequently, the mean time $\overline{T}$ required to compute and establish a new set of forwarding rules and deliver the first packet of the related new traffic flow data to the destination may be structured into 9 components:

1. propagation time between TH_S and GS_S for data packets;
2. propagation time between GS_S and LE_S for data packets;
3. propagation time between LE_S and GE_S for flow entry request;
4. propagation time between GE_S and NM_S for policy and network statistics request;
5. propagation time between NM_S and GE_S for policy and network statistics reply;
6. propagation time between GE_S and LE_S for flow entry reply;

7. propagation times of all ISLs from LE_S to LE_D for data packets;
8. propagation time between LE_D and GS_D for data packets;
9. propagation time between GS_D and TH_D for data packets;

The values of all these components depend on the distances between the involved node pairs. The distances among adjacent LEO satellites belonging to the same orbit is almost constant, therefore the propagation delay of ia-ISLs may be considered as a constant (part of component 7). The distances among GEO satellites and related NMCCs is constant, and, opportunely setting NMCCs positions, is the same for each of the three pairs (GEO, NMCC). Therefore the propagation delay of the (GEO, NMCC) link is constant (components 4 and 6). The distances related to the other elements range between a minimum and a maximum value. In this way, we can estimate the possible range of $\overline{T}$ from a lower to an upper bound considering, respectively, the minimum and maximum values of these distances, and, consequently, the minimum and maximum values of the related propagation delays. Hence, $\overline{T}$ can be obtained as in Equation (4.1). All variables are defined in Table 4.2.

$$\overline{T} = 2 \cdot \left(\overline{t_{TH}^{GS}} + \overline{t_{GS}^{LE}} + \overline{t_{LE}^{GE}} + t_{NM}^{GE} \right) + \sum_{h=1}^{N_L} \sum_{k=1}^{N_L} q_{h,k} \cdot \left[H_{h,k}^{ia} \cdot t_{LE}^{LE} ia + H_{h,k}^{ie} \cdot \overline{t_{LE}^{LE} ie} \right] \quad (4.1)$$

where:

$$t_{GS}^{LE} min \leq t_{GS}^{LE} \leq t_{GS}^{LE} max \quad (4.2)$$

$$t_{GS}^{LE} min = \frac{d_{GS}^{LE} min}{c} = \frac{h_{LE}}{c} \quad (4.3)$$

$$\begin{aligned} t_{GS}^{LE}max = \frac{d_{GS}^{LE}max}{c} = -\frac{R_E}{c} \cdot \sin\theta_{GS}^{LE}min + \frac{1}{c}\cdot \\ \cdot\sqrt{h_{LE}^2 + 2 \cdot h_{LE} \cdot R_E + R_E^2 \cdot \sin^2\theta_{GS}^{LE}min} \end{aligned} \tag{4.4}$$

$$t_{LE}^{GE}min \leq t_{LE}^{GE} \leq t_{LE}^{GE}max \tag{4.5}$$

$$t_{LE}^{GE}min = \frac{d_{LE}^{GE}min}{c} = \frac{h_{GE} - h_{LE}}{c} \tag{4.6}$$

$$\begin{aligned} t_{LE}^{GE}max = \frac{d_{LE}^{GE}max}{c} = -\frac{R_E + h_{LE}}{c} \cdot \sin\theta_{LE}^{GE}min + \frac{1}{c}\cdot \\ \cdot\Big[h_{GE}^2 + 2 \cdot h_{GE} \cdot R_E - (h_{LE}^2 + 2 \cdot h_{LE} \cdot R_E) \cdot \cos^2\theta_{LE}^{GE}min + \\ +R_E^2 \cdot \sin^2\theta_{LE}^{GE}min\Big]^{1/2} \end{aligned} \tag{4.7}$$

$$t_{NM}^{GE} = \frac{d_{NM}^{GE}}{c} = \frac{R_E}{c} \cdot \left[\sqrt{\left(\frac{h_{LE} + R_E}{R_E}\right)^2 - \cos^2\theta_{NM}^{GE}} - \sin\theta_{NM}^{GE}\right] \tag{4.8}$$

$$\theta_{NM}^{GE} = \arctan\left[\frac{\cos(\varphi_{GE} - \varphi_{NM}) \cdot \cos\xi_{NM} - 1512}{\sqrt{1 - \cos^2(\varphi_{GE} - \varphi_{NM}) \cdot \cos^2\xi_{NM}}}\right] \tag{4.9}$$

$$t_{LE}^{LE}ia = \frac{d_{LE}^{LE}ia}{c} = 2 \cdot \frac{R_e + h_{LE}}{c} \cdot \sin\left(\frac{\pi}{S_L}\right) \tag{4.10}$$

$$t_{LE}^{LE}ie\,min \leq t_{LE}^{LE}ie \leq t_{LE}^{LE}ie\,max \tag{4.11}$$

$$t_{LE}^{LE}ie\,min = \frac{d_{LE}^{LE}ie\,min}{c} = 2 \cdot \frac{R_e + h_{LE}}{c} \cdot \sin\left(\frac{\pi}{2 \cdot S_L}\right) \tag{4.12}$$

$$t_{LE}^{LE}ie\,max = \frac{d_{LE}^{LE}ie\,max}{c} = 2 \cdot \frac{R_e + h_{LE}}{c} \cdot \sin\psi \tag{4.13}$$

$$\psi = \arccos\left[\cos\left(\frac{\pi}{S_L}\right) \cdot \cos\left(-\frac{\pi}{P_L}\right)\right] \tag{4.14}$$

Equations (4.4) and (4.7) are obtained from Equation (2.2) in [62], Equation (4.8) is defined in [63], Equation (4.9) is reported in [64], and Equations (4.10), (4.12), (4.13), and (4.14) are obtained from simple trigonometric computations.

$\overline{t_{TH}^{GS}}$	mean propagation delay of TH÷GS links
$\overline{t_{GS}^{LE}}$	mean propagation delay of GS÷LE links
$\overline{t_{LE}^{GE}}$	mean propagation delay of LE÷GE links
t_{NM}^{GE}	propagation delay of NM÷GE links
N_L	overall number of LEO satellites
$q_{h,k}$	probability that LE_S and LE_D are the h^{th} and k^{th} LEO satellite respectively
$H_{h,k}^{ia}$	number of ia-ISL in the path between LE_h and LE_k
$t_{LE}^{LE}ia$	propagation delay of ia-ISLs
$H_{h,k}^{ie}$	number of ie-ISL in the path between LE_h and LE_k
$\overline{t_{LE}^{LE}ie}$	mean propagation delay of ie-ISLs
d_{GS}^{LE}	distance between ground stations and LEO satellites
c	speed of light in vacuum
h_{LE}	LEO satellite altitude
R_E	mean Earth radius
$\theta_{GS}^{LE}min$	minimum elevation angle between ground stations

	and LEO satellites
d_{LE}^{GE}	distance between LEO and GEO satellites
$\theta_{LE}^{GE}min$	minimum elevation angle between LEO and GEO satellites
h_{GE}	GEO satellite altitude
d_{NM}^{GE}	distance between NMCCs and GEO satellites
θ_{NM}^{GE}	elevation angle between NMCCs and GEO satellites
φ_{GE}	GEO satellite longitude
φ_{NM}	NMCC longitude
ξ_{NM}	NMCC latitude
$d_{LE}^{LE}ia$	distance between two LEO satellites linked through an ia-ISL
S_L	number of LEO satellites per orbital plane
$d_{LE}^{LE}ie$	distance between two LEO satellites linked through an ie-ISL
P_L	number of LEO satellite orbital planes

Table 4.2: List of defined variables

4.7 Results and Final considerations

To perform computations we use the python library *math*. To model the considered LEO satellite constellation we exploit the python library *networkx* that allows the creation of a connected graph and the association of weights to its edges. In our case, the weights represent the communication delays of the ISLs.

To the purpose of computing the mean overall time to establish the routing path between TH_S and TH_D, we need to set the position of the control centre defined in terms of latitude and longitude coordinates, the altitude of geostationary satellites and their positions in terms of longitude. The longitude of the NMCCs and GEO satellites are always considered equal ($\varphi_{GE} = \varphi_{NM}$), so that the elevation angle between them, reported in Equation 4.9, depends only on the latitude of the NMCCs. Other parameters to be set for the experiments are $\theta_{GS}^{LE}min$ and $\theta_{LE}^{GE}min$, which, respectively, are the minimum elevation angle between a ground station and a LEO satellite and the minimum elevation angle between a LEO and GEO satellite, respectively. These parameters are set to a value of $10°$ in order to avoid the presence of obstacles inside the communication path between the involved entities.

To take into account the effects of different paths in the simulations, the parameters $q_{h,k}$, $H_{h,k}^{ia}$, and $H_{h,k}^{ie}$, defined in Table 4.2, are introduced. Assuming each couple of LEO satellites (h, k) having the same probability of being the end points of the routing path's satellite portion, we assign the value of $\frac{1}{N_L^2}$ to the parameter $q_{h,k}$. Concerning the number of hops in the considered paths, we derive the values by using the build-in function of the python library *networkx* which returns all the possible paths between two nodes in a selected graph, thus allowing to compute both $H_{h,k}^{ia}$ and $H_{h,k}^{ie}$.

The set of parameters used in the simulations are reported in Table 4.3.

Figure 4.7 shows the minimum and maximum values of $\overline{T}$ with respect to the LEO satellite altitude h_{LE} which influences the number of satellites and orbital planes considered in the model, as reported in [62]. Since the aim is that LEO satellites completely cover the Earth surface independently of their altitude, a specific number of orbital planes and satellites is needed: Table 4.4 shows the number of orbital planes and LEO satellites necessary in

Parameter	Value
R_E	6371 [km]
h_{GE}	35790 [km]
h_{LE}	[400 – 1500] [km]
ξ_{NM}	45°
$\theta_{GS}^{LE}min$	10°
$\theta_{LE}^{GE}min$	10°
$q_{h,k}$	$\frac{1}{N_L^2}$
c	299792.458 [km/s]

Table 4.3: Set of initial parameters used for the experiments

order to cover the entire Earth with respect to the altitude of LEO satellites. From Table 4.4 we can notice that increasing the altitude, the number of orbital planes and satellites per orbital plane decreases.

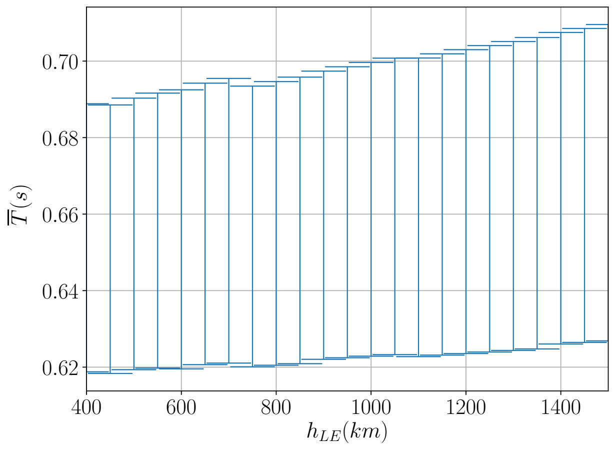


Figure 4.7: Minimum and maximum $\overline{T}$

Altitude $[km]$	S_L	P_L
400	15	8
450	14	7
500	13	7
550	13	7
600	12	6
650 - 700	11	6
750 - 850	10	5
900 - 1050	9	5
1100 - 1350	8	4
1400 - 1500	7	4

Table 4.4: Number of P_L and S_L needed for the complete coverage of the Earth

In Figure 4.7, as expected, $\overline{T}$ globally exhibits an ascending trend, since the end-to-end-delay grows with the satellite altitude. We can also notice that, every time there is a decrease in the number of orbital planes, and, consequently, a reduction of the number of satellites involved in the complete Earth coverage, $\overline{T}$ slightly decreases. This reduction is due to the fact that a lower number of satellites is needed in order to obtain a complete coverage of the Earth and, as a consequence, the number of hops to be performed by the flows in order to reach the destination decreases. In any case, this small decrease is balanced when the distance between two satellites increases due to the growing altitude. Consequently, the behaviour of $\overline{T}$ keeps an ascending trend. It is worth noting that, considering a maximum altitude equal to $1500km$, the maximum end to end delay is about $0.709s$. This type of computation is essential to decide if a future 5G service can be forwarded to satellite portions or not. The time required to route and forward traffic over the considered SDN-based satellite network is not excessive but, of course, is in contrast with demanding delay sensitive applications.

Traditional and currently operating satellite communication networks often rely on ad-hoc hardware components and proprietary software solutions. This hinders the integration of satellite and terrestrial networks and also of different satellite networks. New solutions are arising to ease the integration and allow satellites to be part of an overall Internet composed of heterogeneous networks, according to the network evolution envisioned in the 5G environment. The employment of SDN in satellite networks is not straightforward but could be a possible solution.

4.8 Conclusion about the role of SDN in the Satellite environment

Satellite communication networks are going to have a crucial role in the 5G ecosystem which can take advantage of their high coverage and broadcast capability to increase the number of networked users, and to improve the reliability and availability of the overall network in particular in cases of emergency and critical missions, service continuity and multimedia distribution. However, their integration with 5G terrestrial networks is a non-trivial task and entails evolutions of the current structures. From the networking viewpoint, network virtualization is a concept that will bring benefits in terms of lower costs, higher flexibility, and tailored service provision. The adoption of SDN and NFV technologies into the satellite domain is seen as a key element to accomplish satellite and mobile terrestrial networks integration, allowing the creation of a heterogeneous 5G network architecture and the provision of dedicated slices. In this vision, satellite network architectures should be augmented with autonomous and flexible management of service lifecycle operations, including the real-time monitoring of performance and other 5G KPIs.

In this chapter we surveyed the outputs of some the main research projects and studies about the integration of satellite networks in the 5G environment, with the purpose of highlighting the current status of the research in this field. We have described the open issues to be investigated before defining and standardizing an SDN/NFV-based solution for satellite networks. Considering the difficulties of virtualizing these networks, an architectural framework and a possible road-map including a set of possible future steps to allow a gradual virtualization starting from the satellite ground infrastructure up to on-board functionalities have been proposed. Furthermore, we investigates the employment of the SDN paradigm in an integrated terrestrial-satellite network where three GEO satellites act as SDN controllers and the Data Plane is embedded in a LEO satellite constellation. A model to estimate the mean time required to complete the SDN control actions and to deliver the first packet of a new traffic flow is also proposed. The obtained results allow understanding which is the lower bound of the required mean delivery time. This value should be used by a user/application to fix the performance requirements in case the satellite component is used or by the network management to decide if traffic may be forwarded through the satellite in a software-defined integrated terrestrial-satellite network. In the proposed time estimation model, we have only considered propagation delays and assumed there is no handoff, i.e. the topology is unchanged during the traffic flow data delivery. Even if the assumptions seem reasonable for short flows, the impact of transmission, queuing and forwarding delays must be considered for wider and possibly congested scenarios. The aspects neglected here will be investigated in future research. Other open issues to be tackled are: alternative integrated terrestrial-satellite network architectures, including, for example, Medium Earth Orbit (MEO) satellites; the study of a different distribution of the SDN Planes inside the network nodes; the impact of the overhead introduced by using the OpenFlow protocol for communication between SDN controller and switches; the synchronization problems due to the rapid movement of satellites; and the issues related to the migration of LEO satellites from one controller to another.

Chapter 5

The Problem of Security

Important applications such as e-business, e-banking, public health service, and defense system control are dependent on computer networks. For this motivation they are often object of attacks by malicious software (malware). Malware is software designed to intrude a computer system without the consent of the owner through the use of viruses, backdoors, spywares, trojans, keyloggers, botnets, and worms [65]. In this context accurate malware detection is a necessity [66]. Countermeasures may be dedicated to specific devices, as happens in the context of mobile devices [67, 68, 69] and FM radios [70], to specific applications such as Internet chats [71], to operating systems such as Android [72], and to given environments such as Delay Tolerant Networks (DTNs) [73] and AODV-Based MANETs [74].

In general Intrusion Detection Systems (IDS) may help tackle malicious intrusions. An IDS is a hardware/software designed to automatically alert when someone or something is trying or has tried to compromise information systems through malicious actions. [75, 76] contain a detailed and interesting classification of Intrusion Detection Systems depending on: the location of the IDS (host based, network based, and hybrid); the detection time (on and off line); the environment (wireless, wired, and heterogeneous); as well as the architecture (centralized/distributed); and the reaction (active/passive). As far as this paper is concerned, the most important IDS classification proposed in [75, 76] regards the processing method adopted to detect possible intrusions: Misuse Detection and Anomaly Detection. Misuse detection defines an abnormal behavior and considers all the other behaviors as normal. Anomaly Detection fixes the normal behavior and considers all the other behaviors as abnormal. From the operative viewpoint the former contains: signature based, rule based, state transition algorithms, and data mining. The latter includes: statistical, distance, profile, and model-based schemes. Misuse Detection (MD) needs to open and inspect the content of the packets or files traversing the IDS either to collect and compare signatures with the available signatures in a malware database or to apply a given set of rules. MD is often very efficient, its drawback stands in the weakness of signatures/rules, which may be referred to dated attacks, and in the required computation time because each single packet needs to be inspected. Anomaly Detection, and, in particular, concerning this paper, Statistical Analysis Based Intrusion Detection (SABID) would like to avoid these drawbacks also at the cost of a lower detection accuracy. Packets could not be opened and inspected and each traffic flow can be monitored over time by measuring the statistics of a set of variables (called features) to distinguish between anomalies (possible malware) and normal behaviors (normal, not infected, traffic). Some more detail about these aspects will be provided in the next Section. In the framework of SABID systems this paper proposes a novel network-based IDS, called SF-IDS (Statistical Fingerprint-IDS). SF-IDS uses the typical flow definition at IP (Internet Protocol) layer and is aimed at deciding whether an IP flow is malware-affected or not. It is structured into a training phase developed by using a ground truth of known flows and an operative classification and decision phase. Both training and classification/decision phases are based on the definition and extraction of a group of statistical parameters related to each IP flow, which represent the Statistical Fingerprint of the flow and on machine learning-based classifiers devoted to distinguish normal from malicious traffic.

5.1 State of the art

5.1.1 Machine Learning-based Classifiers

Machine learning-based classifiers are aimed at identifying to which set of categories a new sample belongs on the basis of a training set composed by data whose category is known. In our case classifiers are used to discriminate normal from malicious traffic as explained in Section 5.2. Machine learning-based Classifiers may be structured into two families: supervised and unsupervised. Supervised classifiers require a training phase during which a number of samples whose classification is known are used to carve N decision regions in the features space, being N the number of the classes to be identified. All the samples whose vectors lie in the same decision region belong to the same class. A sample whose classification is unknown is classified by determining the decision region where the feature vector of the sample falls. The methodology to carve the decision region depends on the chosen algorithm.

Naive Bayes [77, 78, 79], among many others, belongs to the group of Bayesian Classifier [80] and requires the independence of the features. Support Vector Machine (SVM) [78] is a family of methods that, given a set of training samples, each marked as belonging to one of two classes, build a model which assigns new samples to one class or to the other. An SVM model is a representation of the samples as points in space. The two classes must be divided by a gap which should be as wide as possible. New samples whose classification is unknown are assigned to a class depending on which side of the frontier they fall in. The gap may be created in different ways so giving origin to Linear, Quadratic, Cubic, and Radial Basis Functions SVM. K-Nearest Neighbors (K-NN) [77] input consists of the K closest training samples in the feature space. A sample is classified by a majority vote of its neighbors, with the object being assigned to the most common class among its K nearest neighbors. In other words, K-NN uses a reference cell such as an hyper-sphere. The cell is expanded up to include K training samples. In the hyper-sphere case they are the K samples with minimum Euclidean distance. The sample under exam is assigned to the class whose training samples among the K samples are more numerous than the samples of the other classes. DTNB [81] is a simple Bayesian ranking method that combines naive Bayes with induction of decision tables. Ridor [82] models large data sets, which results in rule sets having minimal inter-rule interactions and simple to be maintained. SMO implements the sequential minimal optimization algorithm [83] to train a support vector classifier: training a support vector machine requires the solution of a very large Quadratic Programming (QP) optimization problem. SMO divides the QP problem into a series of smallest possible QP problems that are solved analytically. J48 is a decision tree algorithm that generates a pruned or unpruned decision tree by using C4.5 algorithm [84]: decision tree algorithms begin with a set of cases, or examples, and create a tree data structure that can be used to classify new cases. Each case is described by a set of attributes (or features) which can have numeric or symbolic values. There is a label representing the name of a class associated with each training case. Each internal node of a decision tree contains a test, the result of which is used to decide what branch to follow from that node. The leaf nodes contain class labels instead of tests. In classification mode, when a test case (which has no label) reaches a leaf node, C4.5 classifies it using the label stored there. JRIP implements the propositional rule learner Repeated Incremental Pruning to Produce Error Reduction (RIPPER), proposed in [85]. Mentioned C4.5 and RIPPER operate in two stages. First they induce an initial rule set and then they refine it using a rather complex optimization stage that discards (C4.5) or adjusts (RIPPER) individual rules to make them work better together. PART [86] exploits the fact that rule sets can be learned one rule at a time, without any global optimization, and infers rules by repeatedly generating partial decision trees. Random Tree, authored by Eibe Frank and Richard Kirkby, builds a tree that considers K randomly chosen attributes at each node. It does not perform any pruning and it has an option to allow an estimation of class probabilities (or target mean in the regression case) based on a hold-out set (backfitting). Again authored by Eibe Frank, RBF Network [87] is a fully supervised machine learning scheme that uses Gaussian Radial Basis Function (RBF) Networks. Random forests [88] is an ensemble learning method for classification, regression, and other tasks. It operates through a multitude of decision trees at the training time. Each user is assumed to know about the construction of single classification trees. To classify a new object from an input vector, the input vector is put down each of the trees in the forest. Each tree gives a classification and the tree "votes" for a class. The forest chooses the classification having the most votes over all the trees in the forest. "Random Forests" is a trademark of Leo Breiman and Adele Cutler.

Unsupervised classifiers are aimed at framing the flows under exam within clusters without any "a priori" information about the samples. For the purpose of this research we focused only on supervised classifier but future investigation can be made using unsupervised ones.

5.1.2 Misuse and Statistical Analysis Based Intrusion Detection Systems

A rough comparison about processing method, accuracy, complexity, speed, and limitations between MD and SABID (considered representative of the entire class of Anomaly Detection for the aim of this work) methods is reported in Table 5.1. In practice the comparison is between intrusion detection systems that require the inspection of packets/files/codes and systems based on the analysis of statistical profiles.

Concerning the large and heterogeneous family of Misuse Based Intrusion Detection Systems, recent research includes the following papers, among

	Misuse Based Intrusion Detection	**Statistical Analysis Based Intrusion Detection**
Processing method	It examines the whole packet for signatures/rules	It examines samples of traffic statistically
Accuracy	High	Low
Complexity	High	Low
Speed	Slow	Fast
Limitations	It cannot detect new virus or encrypted flow	A training data set is involved

Table 5.1: Misuse Based Intrusion Detection versus Statistical Analysis Based Intrusion Detection systems.

many others. [89] is a paper whose experimental results show the detection ability of the system to learn effective rules from repeated presentations of a tagged training set. Best system accuracy is close to 90%.[90] develops an automatic categorization system to automatically group phishing websites or malware samples by using a cluster ensemble. Malware categorization results range between 86% and 91%. [91] proposes a host-rule-behavior-based detection method, composed of a clustering engine that groups the objects (e.g., processes and files) of a suspicious program together into a cluster. Obtained results vary depending on the fixed threshold of false positives (see Table 5.4 for a definition): if you want no false positives, then the system can assume

71% true positives but if you relax the threshold you can get true positives rates above 90%: 93.2% with 9.8% false positives and up to about 97% with 22.5% false positives. The authors show that their results are more satisfying than the ones got by commercial antivirus software. Concerning the search and analysis of opcodes (from operation code, a portion of a machine language instruction that specifies the operation to be performed), we can mention [92] and [93]. [92] is aimed at individuating a subset of opcodes suitable for malware detection through SVM (Support Vector Machine). Using opcode sequences typically needs to label a large amount of both malicious and benign code. [93] proposes a method that uses single-class learning to detect unknown malware families. Specific results vary if labeling is performed through malicious or benign software but in general: labeling 60% of the legitimate software assures about 85% accuracy. Among signature-based approaches: [94] classifies packed and polymorphic malware through a fast application-level emulator; the effectiveness is validated by showing that malware is detected as a variant of existing malware in 88% of cases. Classification is also quite quick: 1.3 $[s]$ for a sample set. [95] compares the performance of the intrusion detection systems Suricata and Snort. The percentage of alerts detected is close to 100% for Snort while the one for Suricata heavily varies on the operating network speed: 98% at $1\,Gbps$, 91.8% at $1.5\,Gbps$ and 66.8% at $2.0\,Gbps$. [96] selects the possible signatures and uses only a subset of the necessary ones.

Concerning the systems that use Anomaly Detection (or also hybrid Statistical Analysis/Misuse Detection): [97] proposes a hybrid IDS combining packet header anomaly detection (PHAD) and network traffic anomaly de-

tection (NETAD). The combined action seems to work even if it is difficult to detect precise percentages from the reported results. [98] introduces a hybrid intrusion detection system that combines k-Means and two classifiers: K-nearest neighbor and Naive Bayes for anomaly detection. The goal in [98] is to decrease the false alarm rate when intrusions are detected and classified in 4 categories: Denial of Service (DOS), U2R (User to Root), R2L (Remote to Local), and Probe. The accuracy varies depending on the attack: from 92% of U2R to more then 98% for Probe. [99] describes a two stage architecture to tackle intrusions. In the first stage a probabilistic classifier is used to detect potential anomalies in the traffic. In the second stage a HMM (Hybrid Markov Model) traffic model is used to narrow down the number of IP addresses carrying the attack. The performance depends on the number of states used for the HMN and on other used features: the best configuration provides an accuracy close to 97% and a false alarm rate below 3%. [100] introduces a hybrid detection framework combining misuse detection, which uses a Random Forest classification algorithm, and anomaly detection, which exploits the weighted k-Means scheme. The detection rate of the combined approach is about 98% with a false positive rate of about 1%.
As far as Statistical Analysis Based Detection, two papers are particularly meaningful for the topic of this research, even if they are not strictly related to malware detection: [101] and [102]. Both contributions are aimed at detecting application-layer tunnels throughout Statistical Fingerprints. [101] presents a statistical classification mechanism called Tunnel Hunter devoted to recognize a generic application protocol tunneled on top of HTTP or of SSH. The accuracy is 100% for HTTP tunnels and above 99% for SSH ones.

[102] aims at detecting DNS tunnels. The accuracy for a mix of applications is close to 99%. Another important paper concerning the approach followed in this work is [103], where streaming content changes are detected only through traffic patterns built from the traffic volume achieved by routers. Other articles must be mentioned as relevant for this work. [104] introduces a scheme for intrusion detection operating in WEKA, used also in our research. [105] proposes to structure Machine-Learning-based intrusion detection systems into Artificial Intelligence based and Computational Intelligence based ones. The former refer to the methods from domains such as statistical modeling (as we done), whereas the latter include methodologies such as genetic algorithms, artificial neural network, fuzzy logic, and artificial immune systems. [106] extracts a long list of features from the used dataset [107] and compares, as done in this thesis (Table 5.5), different classifiers such as, among the others, DTNB, JRIP, PART, RIDOR, all providing about 95% accuracy, and SMO, assuring an accuracy above 97%. [108] compares J48, Random Forest and Random Tree in the same operating environment by using the same dataset and list of features presented in [106] and proposes to use a combination of classifiers to enhance the performance, which is above the 99% in the best cases. The obtained results of these classifiers are very similar to the ones shown in Table 5.5. [109] proposes a selection of features by using swarm intelligence algorithms, such as Artificial Bee Colony (ABC) or Particle Swarm Optimization (PSO), and evaluates the performance through the same dataset used in [107].

5.2 Statistical Fingerprint - Based Intrusion Detection System - SF-IDS

5.2.1 Key Ideas

Our idea shares with [101], [102], [110] and the other papers mentioned in the previous section concerning SABID, the idea of detecting something by using statistical analysis. For instance "looking at simple statistical properties of protocol messages, such as statistics of packet interarrival times and of packets sizes" [102] may be useful to perform monitoring actions. "The key idea is that the information carried by packets at the network layer, such as packet-size and inter-arrival time between consecutive packets, are enough to infer the nature of the application protocol that generated those packets" [101]. This sentence, referred in [101] to tunnels, may be literally applied to malware and for this reason useful for our purposes. We think that, observing the statistical features of a specific IP traffic flow, we can get information about the malicious (or not) nature of this flow. We identify an IP traffic flow with the 5-tuple composed of the following fields of the IP and TCP/UDP headers:

- IP Source Address (IP SA)
- IP Destination Address (IP DA)
- TCP/UDP Source Port
- TCP/UDP Destination Port
- Protocol

These fields are considered as two-way (the inversion of Source and Destination Ports and Addresses is considered as one single flow) in the tests in Section 5.4. This choice reduces the number of flows per each trace recorded from the network, and allows creating longer flows that are more robust to the noise than default short flows created automatically by hosts connected to the Internet. The field Protocol defines the protocol used in the data portion of the IP datagram. In practice it specifies the content of the IP packet information field. The Internet Assigned Numbers Authority maintains a list of IP protocol numbers which was originally defined in [111] and are now defined through an online database specified in [112].

5.2.2 SF-IDS Architecture

The overall architecture of the proposed IDS is depicted in Figure 5.1, which refers to a general-purpose architecture to analyze traffic flows whose operative steps are detailed in the following. Such architecture has not been implemented and used to perform experiments for which we have applied a subset of the components appearing in Figure 5.1 and, in particular, the "Packet analyzer" (object of Subsection of 5.2.3), "Malware DB" (used for the training phase) and "Syslogger" (output of the packet analyzer). The practical implementation of the overall architecture in Figure 5.1 is one of the next steps of this research activity.

Packets from/to the Internet traverse the external interface (typically an ADSL/ATM interface) of the system and are processed by a router in order to be properly forwarded. At this level, if needed, some virtual interfaces may be attached to allow sending/receiving packets through tunneling protocols

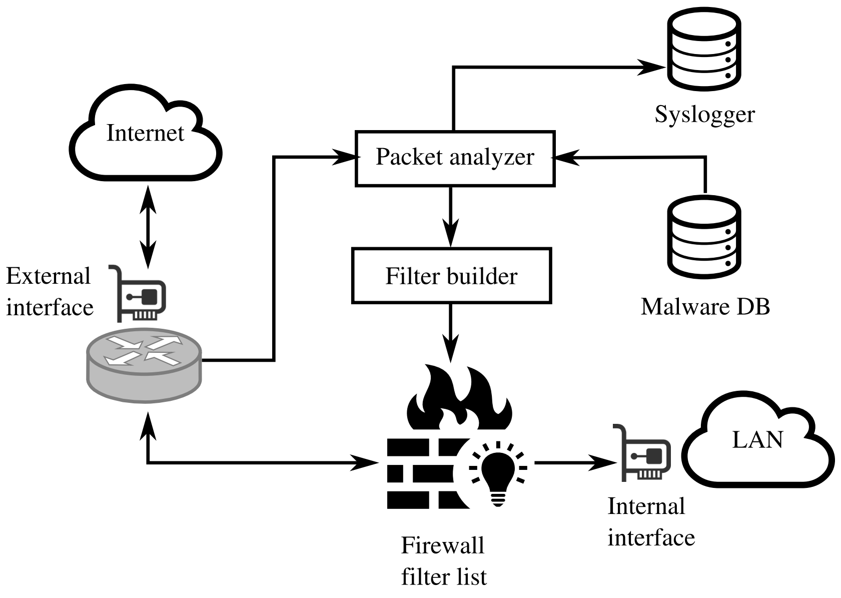


Figure 5.1: SF-IDS overall architecture.

and/or traffic encryption. This allows establishing virtual point-to-point (secure) connections with the aim of creating Virtual Private Networks (VPNs). Thus, different user sites may appear to be part of a unique wide network. User applications, running in different locations, are enabled to communicate with each other and, possibly, share common resources in the same way applications are hosted on co-located computers. However, it is worth noting that the adoption of VPNs only partially reduces the hazard to be victim of malware. Whether a PC in the user LAN is infected by a malware, the PC may easily infect other systems present in the LANs and belonging to same VPN infrastructure. In other words, IDS mechanisms need to be adopted even in presence of secure and encrypted links (tunnels) connecting different sites of the same enterprise. Furthermore, some malicious tunnels may represent the media through which infected packets may be conveyed, as mentioned in the previous section through the proper references. The IP packets traversing the router are inspected by a "Packet Analyzer (PA)" in order to detect possible harmful flows. The PA exploits the features of a set of malwares stored in the so-called "Malware Data Base (MDB)". Whenever a malicious flow is detected, its features are logged in the "Syslogger" subsystem and, correspondingly, a new rule is compiled by the "Filter Builder (FB)" and then added to the filter list ("Firewall Filter List - FFL") of the firewall. The new rule aims at blocking the just detected malicious flow, thus preventing the related malware to access the LAN through the Internal Interface (commonly, an Ethernet interface). It should be highlighted that, if the system has more than one internal interface, each interface has its own FFL.

5.2.3 SF-IDS Packet Analyzer

The Packet Analyzer represents the most original part of our work and the object of the performance evaluation. Figure 5.2 sketches its main components.

Incoming packets feed the "Feature Extractor" (FE) which performs two principal operations. The first one consists in the identification of a traffic flow on the basis of the flow definition provided before. The second operation is the extraction of a number of features, discussed in the next Subsection, from each flow. Under this perspective, each flow is uniquely described by the vector V_f of its features. V_f is the Statistical Fingerprint of the flow. The vector V_f is then passed to a group of classifiers, each of them previously

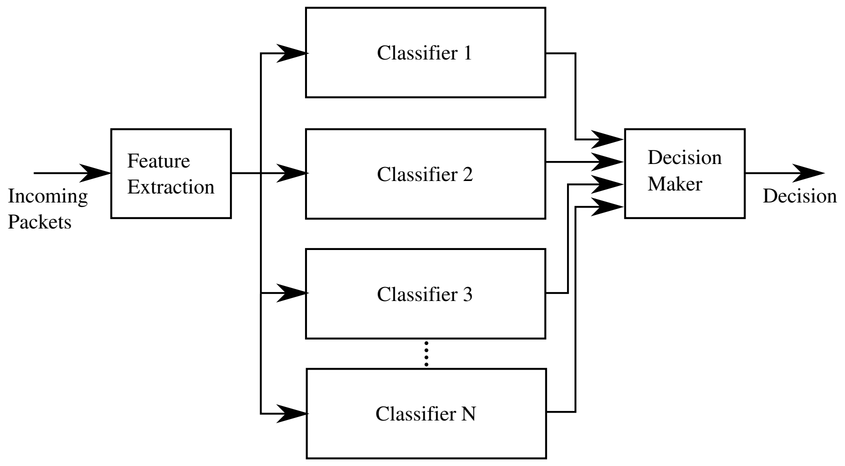


Figure 5.2: Block diagram of the Packet Analyzer.

trained by using known traffic traces to detect the malware presence on the basis of a set of features. Each classifier makes its own decision: the flow is a malware or it is not. Eventually, a Decision Maker merges the decisions made by each classifier in order to produce the final decision regarding the flow under analysis. Strategies adopted to make the final decision are discussed in Subsection 5.4.4.

5.2.4 Statistical Fingerprint: the feature vector

The 14 components (indicated as features) of the vector V_f used to classify each flow are listed in Table 5.2. As said before, the key idea is that the following features associated to each flow are enough to infer the possible malicious nature of the flow. The use of these features is coherent with the literature in the field, see for example [113, 114]. [107] uses a larger and different list, not strictly related to the features of flows. Any change to the features used in the table does not require any substantial change to SF-IDS.

5.2.5 SF-IDS Classifiers

The algorithms used to distinguish normal from malicious traffic on the basis of the set of features extracted from each flow are machine learning-based classifiers. The following supervised classifiers, coherently with the state of the art presented in Section 5.1, have been tested to select the most performant ones.

- Naive Bayes (NB);
- Linear SVM - the frontier between regions is a linear function;
- Quadratic SVM - the frontier between regions is a quadratic function;
- Cubic SVM - the frontier between regions is a cubic function;
- Radial Basis Functions (RBF) SVM;
- K-Nearest Neighbors - K-NN with $K = 1$, and $K = 3$;
- JRIP;
- Random Forest;
- DTNB;
- PART;
- Ridor;
- SMO;

- J48;
- Random Tree;
- RBF Network;

Features	Description
Num_Pack	Number of packets
Tot_Byte_Flux	Number of bytes
Flow_Duration	Duration of the flow in seconds
Byte_Rate	Byte rate
Packet_Rate	Packet rate
Delta_Mean	Average inter-arrival time of packets
Delta_Std	Standard deviation of inter-arrival time
LE	"Entropy" of the packet lengths[2]
DPL	Total number of subsets of packets having the same length divided by the total number of packets of the flow
First_Len	Length of the first packet
Max_Len	Length of the longest packet
Min_Len	Length of the shortest packet
Mean_Len	Average packet length
Std_Len	Standard deviation of the packet length

Table 5.2: Used features for each flow as Statistical Fingerprint.

[2]LE is calculated starting from the normalized occurrences of the packet lengths. Specif-

5.3 Used traffic and performance parameters

5.3.1 Used Malware and Normal Traffic

The tests reported in the performance evaluation have been carried out by downloading traffic samples from [115]. Table 5.3 contains the list of used malwares together with the overall number of analyzed flows and packets. Each flow appearing in Table 5.3 under the label malware is not exclusively composed of malware affected packets but it contains also not affected traffic. Obviously the two components can be distinguished to allow a correct performance evaluation. Table 5.3 includes also the same quantities for the traffic that is not affected by malware and is called "normal traffic". In this case these traces are totally malware free. Each malware has different features.

Cutwail is a botnet aimed at spamming.

Purple Haze is a botnet that records user activities. In practice, it is a keylogger acting at kernel level.

Ramnit is a worm that has been used to get Facebook passwords.

Tbot is a botnet used for DDoS attacks, bank frauds, and cheats by using e-money (bitcoins).

Zeus is a botnet that widespreads a Trojan to infect computers through phishing or false download actions unconsciously performed by users. Its main function is online banking FTP account violation.

ically, being L_i the number of times a packet has a length equal to i, LE is computed as $LE = -\sum_{i=0}^{1526} \frac{L_i}{N} \log_2(\frac{L_i}{N})$, where N is the total number of packets belonging to the flow.

ZeroAccess is a Trojan that affects Microsoft Windows operating systems. It is used to download other malware on the infected machine and is mostly involved in bitcoin mining and click fraud. It may remain hidden on a system by using several techniques.

AlienspyRAT is a Trojan that gives attackers the ability to gain complete remote control of a compromised system. It can be used to collect a range of system-specific data, including operating system version, memory and RAM data, Java version number, and other details, such as passwords, and private information.

Kuluoz is a Trojan that tries to steal passwords and sensitive information. It can also download other malware onto the infected PC.

Sality is a polymorphic file infector. It infects executable files on local, removable, and remote shared drives. It can communicate over a peer-to-peer (P2P) network and has the purpose of relaying spam, compromising web servers, and extruding data.

Normal traffic used in the tests has been captured by using the *tcpdump* utility.

5.3.2 Performance Evaluation Parameters

The performance of each classifier itemized in Subsection 5.2.5 has been evaluated by comparing the results of the classification with the actual class of the flow. Under this perspective, 4 cases, listed in Table 5.4, can occur.

Malware	**Flows**	**Packets**
Cutwail	2347	35674
Purple Haze	7349	324709
Ramnit	25141	155973
Tbot	223	13048
Zeus	202	7443
ZeroAccess	350	2535
AlienspyRAT	1214	9010
Kuluoz	16894	179607
Sality	12939	250784
Normal Traffic	**Flows**	**Packets**
Normal Traffic 1	4969	833368
Normal Traffic 2	12552	3533925
Normal Traffic 3	23351	4428188

Table 5.3: Used malware and normal traffic.

Corresponding quantities in percentage may be defined as $TN = \frac{N_{TN}}{N_N} \cdot 100$, $FP = \frac{N_{FP}}{N_N} \cdot 100$, $TP = \frac{N_{TP}}{N_M} \cdot 100$, $FN = \frac{N_{FN}}{N_M} \cdot 100$, being N_N the overall number of analyzed normal flows, N_M the overall number of analyzed malware affected flows, and N_{TN}, N_{FP}, N_{TP}, and N_{FN} the overall number of True Negatives, False Positives, True Positives, and False Negatives, respectively.

True Positives and True Negatives are the Correct Detection Cases. False Positives (False Alarms) and False Negatives (Missed Detections) are the cases where the system fails, although the impact of FPs and FNs on the

Evaluation Parameter	**Meaning**
True Negative (TN)	A flow is normal traffic, i.e., it is not malware affected and it is correctly classified as normal traffic.
False Positive (FP)	A flow is normal traffic, i.e., it is not malware affected but it is wrongly classified as malware. This case is also called False Alarm.
True Positive (TP)	A flow is malware affected and it is correctly classified as malware.
False Negative (FN)	A flow is malware affected but it is wrongly classified as normal traffic. This case is also called Missed Detection.

Table 5.4: Evaluation Parameters.

overall performance is very different. Consequently the percentage of correct decisions may be evaluated through the parameter Accuracy (Acc) defined as:

$$Acc = \frac{N_{TN} + N_{TP}}{N_{TOT}} \cdot 100 \tag{5.1}$$

where $N_{TOT} = N_N + N_M$ is the overall number of flows.

5.4 Performance evaluation

5.4.1 Tools

The tests have been carried out by using a free machine learning software called WEKA (Waikato Environment for Knowledge Analysis) [116, 117, 118], introduced in 1997 at the University of Waikato, New Zealand, using the 3.6.10 version of the software. WEKA is written in Java, supports standard algorithms for data preprocessing, clustering, classification, regression, visualization, and feature selection. All data have to be available in this format to be analyzed. Detailed WEKA characteristics are reported in [118]. All SF-IDS Classifiers listed in Subsection 5.2.5 are supported by this tool.

5.4.2 Evaluation of Single Classifiers

The analysis has been carried out taking into account the entire set of flows in Table 5.3. Each single trace, both malware and normal, has been divided into two parts (50% of the trace each). The first part of each trace has been used to compose the file for the training phase, the second part to build the file employed for the tests. The goal is to distinguish malware affected flows and normal traffic.

Table 5.5 shows, for each classifier in Subsection 5.2.5, the obtained Accuracy and the percentage of True Positives, False Negatives, True Negatives, and False Positives. The last column of Table 5.5 contains also the 95% Confidence Interval (CI) of the Accuracy values, computed through the known formula $Acc \pm 1.96 \cdot \sqrt{\frac{Acc(1-Acc)}{N_{TOT}}}$, where N_{TOT} is the overall number of flows used for testing, taking the values from Table 5.3, as indicated above. The confidence interval may be simply computed in the same way also for TP, FN, TN, and FP. A good number of classifiers is close to 99% concerning

Classifier	*Acc*	*TP*	*FN*	*TN*	*FP*	*CI*
1-NN	97.21	97	3	97.5	2.5	97.0785÷97.3565
3-NN	97.32	97.2	2.8	97.5	2.5	97.1890÷97.4618
CubicSVM	51.02	99.6	0.4	3	97	50.5976÷51.4428
DTNB	97.45	98.9	1.1	96	4	97.3225÷97.5887
J48	98.03	98.8	1.2	97.3	2.7	97.9186÷98.1532
Jrip	97.86	98.9	1.1	96.8	3.2	97.7407÷97.9851
LinearSVM	88.20	93.6	6.4	82.9	17.1	87.9372÷88.4824
NaiveBayes	89.68	99.7	0.3	79.7	20.3	89.4239÷89.9381
PART	98.06	99	1	97.2	2.8	97.9493÷98.1821
QuadraticSVM	83.63	90.9	9.1	76.4	23.6	83.3272÷83.9526
Random Forest	98.35	99.3	0.7	97.5	2.5	98.2503÷98.4651
Random Tree	97.56	97.7	2.3	97.4	2.6	97.4332÷97.6938
RBFNetwork	75.93	85.2	14.8	66.8	33.2	75.5766÷76.2992
RBFSVM	86.87	90.3	9.7	83.5	16.5	86.5908÷87.1616
Ridor	98.00	99.3	0.7	96.7	3.3	97.8899÷98.1261
SMO	88.19	93.5	6.5	82.9	17.1	87.9259÷88.4713

Table 5.5: Percentage of *Acc*, TP, FN, TN, FP, and *CI* for *Acc* by varying the applied Classifier

the percentage of True Positives (and, in consequence, close to 1% concerning the percentage of False Negatives) but not all of them provide also satisfying results in terms of True Negatives and False Positives. In this view, from the results reported in Table 5.5, it is possible to individuate the most promising classifiers. To this goal we use the metric in Equation (5.2) that minimizes the sum of the percentage of False Positives and False Negatives.

$$min(FP + FN) \tag{5.2}$$

The resulting three best classifiers are: Random Forest, J48, and PART.

5.4.3 Classifier Performance to distinguish Single Malware affected flows and Normal Traffic

The ability of the selected classifiers to correctly detect each single malware has been also checked. Given the training phase operated to get the results in Table 5.5, we have tested the three classifiers by using the 50% of the traces of each malware (which contains, as said before, both affected and not affected packets) not used for training. The performance in terms of *Acc, TP, FN, TN*, and *FP* (as well as the 95% Confidence Interval of the Accuracy) is shown in Tables 5.6, 5.7, and 5.8, for J48, PART, and Random Forest respectively. All selected classifiers offer excellent performance even if Random Forest is slightly more efficient. It perfectly distinguishes Kuluoz, Tbot, and ZeroAccess from Normal Traffic (100% Accuracy) and, in particular, provides an Accuracy of 97.78% for Cutwail. J48 and PART get for the same malware an Accuracy of 91.48% and 90.46%, respectively. The performance of Random Forest is again the best for Purplehaze: 99.95% against 98.53% of J48 and 93.42% of PART. Concerning Zeus, Random Forest and J48 allow getting an Accuracy of 99%, PART of about 94%. The performance for AlienspyRAT and Ramnit is the same for all the considered

J48						
Malware	*Acc*	*TP*	*FN*	*TN*	*FP*	*CI*
AlienspyRAT	99.83	100	0	75	25	99.5127÷100
Cutwail	91.48	100	0	81.7	18.3	89.8853÷93.0789
Kuluoz	99.98	100	0	99.5	0.5	99.965÷100
Purplehaze	98.53	98.9	1.1	88	12	98.1416÷98.9196
Ramnit	96.74	100	0	68.8	31.2	96.4364÷97.0566
Sality	99.56	99.6	0.4	99.6	0.4	99.4072÷99.7272
Tbot	99.10	100	0	99.1	0.9	97.3649÷100
ZeroAccess	98.28	98.2	1.8	100	0	96.3625÷100
Zeus	99.00	100	0	98.5	1.5	97.0789÷100

Table 5.6: Percentage of *Acc*, TP, FN, TN, FP, and *CI* for *Acc* using J48 Classifier

classifiers as well as, substantially, for Sality.

5.4.4 Classifiers Acting in Parallel

A possible alternative to the use of a single classifier is the exploitation of a bank of classifiers as shown in the Packet Analyzer in Figure 5.2. Specifically, a group of different classifiers act in parallel and communicate their decisions to one Decision Maker (DM) block.

The DM block can make the final decision about malware affection or not in different ways. It can state that a flow is malware affected either if at least one single classifier has taken this decision (*Dominant*), or following

PART						
Malware	*Acc*	*TP*	*FN*	*TN*	*FP*	*CI*
AlienspyRAT	99.83	100	0	75	25	99.5127÷100
Cutwail	90.46	97.8	2.2	82.1	17.9	88.7796÷92.1404
Kuluoz	99.94	99.9	0.1	100	0	99.8889÷99.9927
Purplehaze	93.41	93.4	6.6	94	6	92.6131÷94.2169
Ramnit	96.74	100	0	68.8	31.2	96.4364÷97.0566
Sality	99.35	98.7	1.3	99.7	0.3	99.1552÷99.5466
Tbot	99.10	80	20	100	0	97.3649÷100
ZeroAccess	98.85	98.8	1.2	100	0	97.2822÷100
Zeus	94.05	94.1	5.9	94	6	89.4493÷98.6695

Table 5.7: Percentage of *Acc*, TP, FN, TN, FP, and *CI* for *Acc* using PART Classifier

the majority of the decisions of the single classifiers (*Majority*), or if all single classifiers have taken this decision (*Unanimity*). The three classifiers selected before (J48, PART, and Random Forest) have been used to compose the mentioned bank of classifiers acting in parallel. The results about *Accuracy, TP, FN, TN* and *FP* are reported in Table 5.9 together with the 95% Confidence Interval of the Accuracy. Training and testing files are the same as in Subsection 5.4.2. Even if the reported percentage are very similar for all the DMs, some remarks may be made. The "*Dominant*" DM maximizes the percentage of True Positives (and consequently minimizes the percentage of False Negatives) with respect to both the other two DMs (and this

Random Forest						
Malware	*Acc*	*TP*	*FN*	*TN*	*FP*	*CI*
AlienspyRAT	99.83	100	0	75	25	99.5127÷100
Cutwail	97.78	99.5	0.5	95.8	4.2	96.9435÷98.6271
Kuluoz	100	100	0	100	0	100÷100
Purplehaze	99.94	99.9	0.1	100	0	99.8702÷100
Ramnit	96.73	100	0	68.8	31.2	96.428÷97.049
Sality	99.79	99.8	0.2	99.8	0.2	99.69÷99.9082
Tbot	100	100	0	100	0	100÷100
ZeroAccess	100	100	0	100	0	100÷100
Zeus	99.00	100	0	98.5	1.5	97.0789÷100

Table 5.8: Percentage of *Acc*, TP, FN, TN, FP, and *CI* for *Acc* using Random Forest Classifier

is expected) and the most performant classifier Random Forest (see Table 5.5) that assures *TP* and *FN* equal to 99.3% and 0.7% respectively. The same quantities for the "*Dominant*" DM are 99.69% and 0.31%. Even if the difference is not evident, the very slight improvement means that there is an intervention of J48 and PART that detect malware affection when Random Forest (rarely) fails. The improvement in terms of *TP* and *FN* is paid by a worst performance concerning *TN* and *FP*. The decrease of *TN* happens not only with respect to the other DMs (again expected) but with respect to the values got by Random Forest in Table 5.5.

Increasing the necessary number of malware decisions made by single

DM Strategy	*Acc*	*TP*	*FN*	*TN*	*FP*	*CI*
Dominant	98.28	99.68	0.32	96.89	3.11	98.1735÷98.3931
Majority	98.28	99.25	0.75	97.33	2.67	98.1754÷98.3949
Unanimity	97.88	98.04	1.96	97.73	2.27	97.7637÷98.0069

Table 5.9: Percentage of *Acc*, TP, FN, TN, FP, and *CI* for *Acc* obtained at the output of the three DMs

classifiers to assign a flow to the malware class allows reducing the percentage of False Positives (2.67% for "*Majority*" and 2.27% for "*Unanimity*") and, consequently, increasing the percentage of True Negatives (97.33% and 97.73%, respectively for "*Majority*" and "*Unanimity*") at cost of *FN* and *TP*. Similarly as happens for "*Dominant*" about *TP* and *FN*, "*Unanimity*" allows getting *TN* and *FP* values better than the ones got by Random Forest in Table 5.5, again for the intervention of J48 and PART which mitigate the rare erroneous decisions of Random Forest.

5.5 Considerations

In this research we propose a network-based IDS called SF-IDS (Statistical Fingerprint - IDS) devoted to decide whether an IP flow is malware-affected or not. SF-IDS is structured into a training phase and a classification/ decision phase. Both phases are based on the definition and extraction of a group of IP flows statistical parameters that represent the Statistical Fingerprint. The key idea is that the Statistical Fingerprint may help detecting the nature (malicious or not) of each flow. The classification/decision phase consists of a “Feature Extractor” (FE), a bank of classifiers, and a Decision Maker that merges the decisions of the classifiers. The performance evaluation traces a possible streamline in view of a future practical implementation and it is structured as follows.

1) Evaluation of the single classifiers and choice of the best ones. To perform this choice we have adopted as a metric the sum of False Alarms and Missed Detections and we have selected the schemes providing the minimum values. Random Forest is the best one but also J48 and PART provide excellent results. The selected schemes are very efficient even if applied to each single malware. In particular, Random Forest assures a null percentage of False Negatives and False Positives for Kuluoz, Tbot, and ZeroAccess, and a very close to null percentage for Sality and Zeus. Random Forest also assures satisfying results for Cutwail. It is very efficient to recognize AlienspyRAT and Ramnit (100% *TP* - 0% *FN*) but it has some difficulties to identify normal traffic, often interpreted as these malwares (75% *TN* - 25% *FP* for AlienspyRAT and 68.8% *TN* - 31.2% for Ramnit).

2) Evaluation of the scheme including three classifiers (Random Forest, J48, and PART acting in parallel) and a Decision Maker (DM) that makes decisions on the basis of three different strategies: “*Dominant*”, “*Majority*”, and “*Unanimity*”.

The choice among the three DMs depends on a possibly adopted risk function/performance tuning. For example, in this case, the balance between Missed Detections and False Alarms we want to get: the “*Dominant*” DM allows minimizing Missed Detections at cost of False Alarms; increasing the number of algorithms necessary to classify a flow as a malware allows achieving better results in terms of False Alarms but implies a performance decrease as concerns Missed Detections. The minimum percentage of False Alarms is got by "*Unanimity*" DM.

The obtained results open the door to an actual development of the software needed to implement the overall architecture proposed here that will be discussed in the next chapter. For this implementation we chose to use the SDN paradigm, which allows us to take advantage of its unique characteristics.

Chapter 6

SDN IDS Implementation

The number of people accessing the Internet is growing rapidly leading, on one hand, to new possible attacks used by cyber criminals and, on the other hand, to an increased complexity in the network management. It is crucial designing systems able to prevent cyber attacks. At the same time, many efforts are provided in order to get tools that can make easier network management. The Software Defined Networking (SDN) paradigm has been designed with this aim allowing network administrators to manage networks easily. This chapter deals with an original Intrusion Detection System that exploits an SDN architecture to get the information needed to feed a statistical-fingerprint based IDS. Specifically the proposed system collects traffic data suitable to detect the possible presence of malware inside the network, and describes the design and implementation of an application developed upon a SDN controller (Ryu) and its role in the malware detection process.

6.1 Introduction

IDSs [75] may be classified depending on: data source (host based, network based, and hybrid); detection time (on and off line); environment (wireless, wired, and heterogeneous); architecture type (centralized/distributed); reaction (active/passive); and processing (Misuse Detection and Anomaly Detection). We focus the attention on reactive network based systems, possibly operating online over heterogeneous networks. In parallel with the evolution of IDSs, the need of simplifying network management has brought to the development of the Software Defined Networking (SDN) [119, 120] paradigm.

Combining a malware detector IDS and SDN may represent a step forward in the service provided by SDN and may allow simplifying the IDS design by means of SDN functions.

6.2 State of The Art

Although the solution proposed in [121] is not directly linked to malware detection, it introduces a possible approach for collecting flow statistics in SDN. The author explains that the use of SDN is essential to allow deep accuracy and granularity without introducing too much communication overhead inside the network. SDN is able to control both temporal (how often to collect data) and spatial (how deep should the inspection be inside the packet) granularity and to distribute flow counting tasks in a smart way among all the switches in the network.

In [122], the authors design an environment that exploits SDN to implement an IDS for a network of Embedded Mobile Devices, so avoiding the problems of standard IDSs within this kind of network such as the inability to cope with end-host mobility and the limited set of actions which can be taken in response to anomalies. Without specifying any particular anomaly detection algorithm, the authors classify what kind of anomalies can be observed: Stateless Flow, Stateful Flow, Volumetric Anomalies, and Physical Anomalies. Within this classification, the category closer to our approach is the volumetric anomaly, which is revealed from the statistics sent by the switches to the controller. However, the reference [122] focuses on the changes of the overall traffic volume, whereas we consider more specific statistical features of flows.

The authors of [123] propose a behavioral-based Security Monitoring System that exploits the flexibility of SDN to orchestrate the detection system. The used controller is Ryu but the collection of statistical data and the classification of flows are delegated to sFlow that represents an additional element in the network. The used classifier in [123] is SVM, a supervised classification algorithm in line with our design choice.

In [124] a DDoS detector is implemented by SDN to allow recognizing malicious flows without any deep packet inspection. The system architecture is similar to the one proposed in this chapter, even though some differences arise, mainly concerning the classification phase. [124] chooses NOX as a controller and develops an application that collects flows' statistics at predetermined time instants. 6 features are extracted from the collected statistics. Self Organizing Map (SOM), an unsupervised classifier, is employed for flow classification.

6.3 SDN Controller Employed within an IDS

6.3.1 General Description of Ryu

The controller chosen to implement our IDS is Ryu [125], an open source, component-based, software defined networking framework written in python, which provides software components by using API that make easy for developers to create new network management and control applications. The general structure of the code is sketched in Figure 6.1 that highlights how the main central framework, responsible for the whole system management, communicates with the switches through the OpenFlow protocol and with the different apps through the APIs.

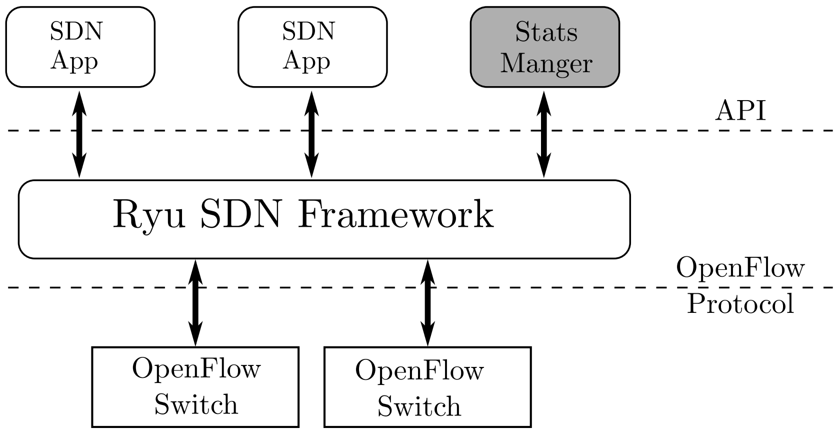


Figure 6.1: Ryu scheme

The main contribution is the design and implementation of an application (app) called `stats_manager`, aimed at managing the flow tables in the used SDN switch, providing the information needed to classify the traffic, and making decisions about "malware/normal" on the examined traffic.

6.3.2 Flow Structure

A traffic flow is usually defined as a group of packets sharing some common characteristics. We use one of the most common conventions according to which a flow is a set of packets having the same 5-tuple: IP source address, IP destination address, TCP/UDP source port, TCP/UDP destination port, Protocol field of IP header. For each flow we store the data in Table 6.1.

Key	Description
IP_src	IP source address
port_src	TCP/UDP source port
IP_dst	IP destination address
port_dst	TCP/UDP destination port
protocol	Protocol field in the IP header
first_len	length of the first packet
pkt_count	number of packets in the flow
byte_count	number of bytes in the flow
dur_sec	duration of the flow(in seconds)
dur_nsec	nanoseconds exceeding dur_sec
byte_rate	byte rate of the flow
pkt_rate	packet rate of the flow
avg_iat	average inter-arrival time between packets
avg_pl	average packet length
state	current state of the flow
extra_p	number of packets not registered in the statistics
extra_b	number of bytes not registered in the statistics
label	class of the flow ('normal' or 'virus')

Table 6.1: Flow Dictionary Structure

The first 5 lines define the flow; the following 9 lines contain the features selected in the previous chapter, which can be computed by the SDN controller directly from the information received by the SDN switch through the statistic reply message; last four lines are better explained in the following:

- **state:** describes the state of each flow with respect to the current *time window* (see section 6.3.3):
  - 'B' means "begun"
  - 'C' means "continued"
  - 'E' means "ended"
- **extra_p and extra_b:** some packets such as the packet-in, i.e. the first packet of a new flow, and corresponding bytes cannot be considered by the SDN switch counters. In other words they cannot be detected by using the statistic reply message. These fields allow the app not to lose this information. These aspects will be elaborated in the following.
- **label:** it is the ground truth about the nature of the flow: normal or affected by malware. During the training phase this information is used to give correct examples to the classifier. In the test phase this field is ignored by the classifier and it is used only to assess the system performance.

Data related to a single flow are stored in the application by two structures: `active_flows`, which contains all the flows currently active in the network, and `ended_flows`, which maintains the information about the flows recently terminated.

6.3.3 Structure of the "Stats_Manager" Application

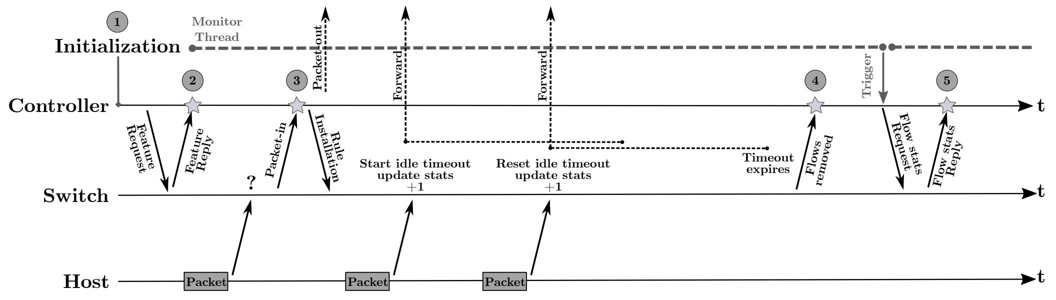


Figure 6.2: General picture of how the app interacts with Ryu

Being Ryu an event-based controller, developing an application for this framework means coding functions that will be executed when a particular event happens. Figure 6.2 depicts the most relevant events for our goal (denoted with a star) and their temporal action. In detail:

1) **Initialization**: this event happens only when the controller instance is created, all the internal variables are initialized, and the `monitor` thread, which will periodically trigger the statistics' request, is started. The controller periodically sends a flow statistic request message in order to gather information about the network traffic from all the switches in the network (just the used one in our case). The time period elapsed between two consecutive messages is called *time window* and it was empirically set to 30 seconds.

2) **Feature Reply**: shortly after the boot, the controller needs to collect some information about the network it has to control. To this purpose it sends a feature request message to the connected switch that, in turn,

answer through a feature reply message, announcing what optional features it supports. Being the first communication between controller and switch, the initialization of flow tables occurs.

3) **Packet-in**: the first packet of a new flow doesn't match any rule in the flow tables, thus it is sent directly to the controller where two kinds of actions are performed: standard packet-in management and start of the IDS statistics collection. The former relates to the tasks that are usually done by every SDN controller: the path for the new flow is computed, the needed rules are installed in the switch, and the first packet is sent back to the sender switch encapsulated in a packet-out message. The latter is strictly related to "Stats_Manager" application: the unknown flow must be recorded in the active flows database and the length of the first packet (first_len in Table 6.1) is stored.

4) **Flow removed**: if no packet matches a given rule for a specified number of seconds, called *idle_timeout*, the flow is considered ended and the rule is removed from the table. When this occurs, the switch sends a packet to the controller, containing the statistics of the removed rule, to notify the event. This is the *asynchronous* way to collect statistics because it may happen at any time instant, as it is strictly dependent on the traffic. The controller extracts the flow identifiers and uses them as indexes to retrieve the specific flow in the active flows database. The measured features (pkt_count, byte_count, dur_sec, dur_nsec, in Table 6.1) are immediately saved and the derived features (byte_rate, pkt_rate, avg_iat, avg_pl) inferred from the previous ones through sim-

ple processing. Finally, the flow entry is removed from the active flow database and inserted in the ended flow database.

5) **Flow stats Reply**: as stated before, every 30 seconds the controller asks the switch for the statistics of the currently active flows. As a reply, the switch sends an OpenFlow packet containing the statistics of all the flows to the controller, as illustrated in Figure 6.3. This is the *synchronous* way to collect statistics since it is regularly scheduled by the controller. Similarly to the asynchronous case, every flow identifier is used as an index to look for the flow entry in the active flows database. Once found, the entry is updated with the new data, before being stored again.

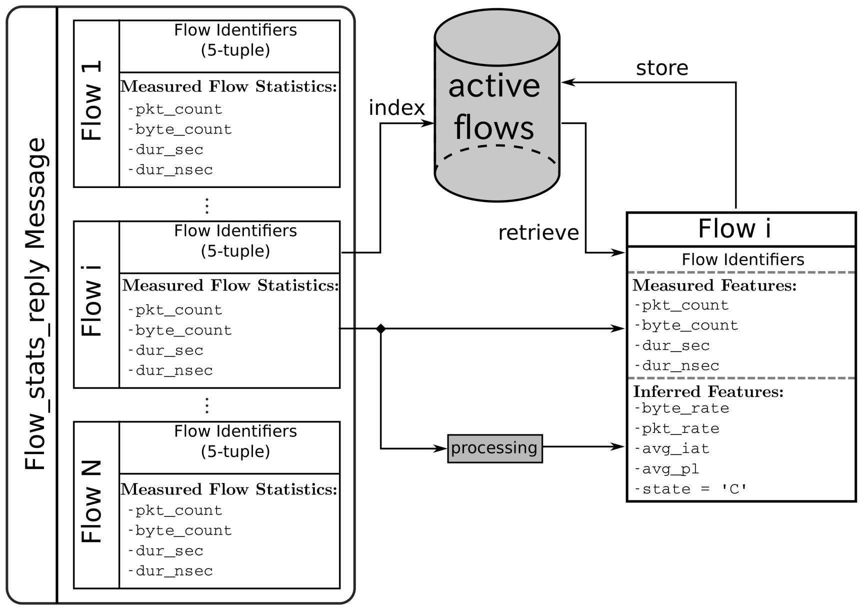


Figure 6.3: Flow Stats Reply

Upon flow stat replies are received at the end of a *time window* additional processing steps are done by the classification process, as shown in Figure 6.4.

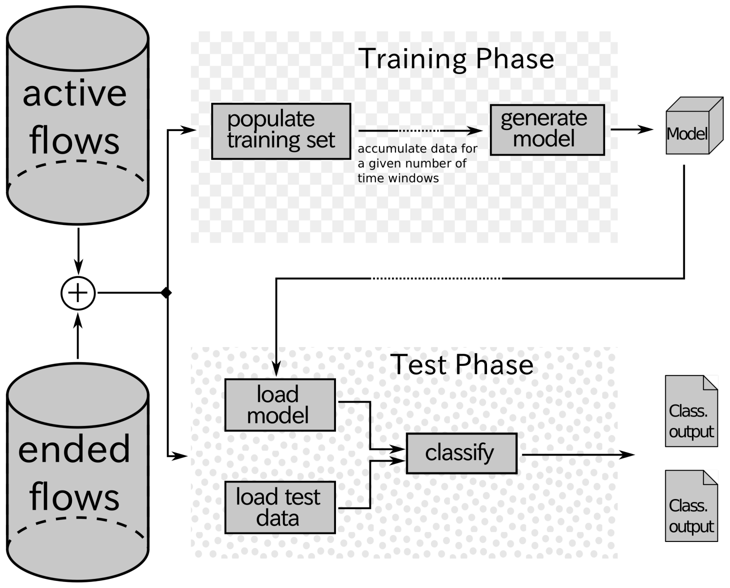


Figure 6.4: Classification Process

All the data related to both active and ended flows are merged to form a unique dataset. The subsequent processing steps depend on the phase the controller is actually performing.

- *Training Phase*: received data are added to the training set of the classifier; when enough data are gathered, they are used to train the classifier. This process produces the classifier's model as a result, which is stored and used in the test phase.
- *Test Phase*: received data are directly addressed to a classifier's module

for the actual classification. Even though the output of a single flow classification is just a binary label stating "malware" or "normal", the system produces two different text files as output.

The first one contains the details of all the analyzed flows and reports all the fields of the flow structure, in Table 6.1, together with the result of the classification process. Consequently it is possible to check how the classifier performs for each single flow.

The second file contains a general report for every time window, containing the total number of flows, the number of normal and malware flows, and the usual classification metrics: true positives, true negatives, false positives, false negatives.

6.4 Used Traffic

In order to test our approach we have to collect traffic information abut malwares. The traces used in this research with the purpose of verifying the proposed architecture, are composed by a mix of different traffic generated by different malwares, which are briefly described in the following:

AlienspyRat: it belongs to the Remote Access Trojan family. Once activated, this software allows collecting system information, updating and downloading other malware. The malware sends captured information to the central server and waits to receive commands.

Asprox: is a spam botnet emerged in 2007. It is known for sending mass of phishing emails used in conjunction with social engineering lures (e.g., booking confirmations, postal-themed spam, etc.). This botnet arrives as an attachment to spammed messages disguised as notifications from postal companies, as well as airline booking confirmations.

Cutwail: is a botnet used to generate spam emails using the contacts in address books. The malware receives instructions from a command and control server about which and how many messages to send. Once it has completed the task, it sends a full report on the number of sent messages and on any found errors to the controller.

Darkness: also called Optima, it is a botnet specialized in DDoS attacks. It waits for commands from a Command and Control (C&C) server that sends encrypted control messages to the infected machines.

Kuluoz: is a botnet aimed at sending phishing emails that simulate messages sent by postal administrations, combined with the use of social engineering techniques. Furthermore, the control server is able to send commands to the infected machines to download and execute pay-per-install programs so to ensure gains to the botnet manager.

Madness: is a distributed denial of service botnet growing in size and popularity. It infects computers running Windows and communicates with its command and control server via HTTP by using a simple client-server model.

Neris: is a botnet that uses an http-based channel to communicate with the C&C server. The main aims of this malware, after establishing a communication with the C&C, are to send spam and perform click-fraud by using advertisement services.

Purplehaze: is a botnet targeted to take the control of machines with the aim of using them to generate many clicks on online advertising sites. It can generate a high volume of traffic on web sites containing advertisements or links in very short time.

Ramnit: this trojan primarily spreads through a contact with infected removable devices, mainly USB flash memory. Once installed, this program connects with a remote server via TCP port 443, sending all the obtained information on the infected machine.

Tbot: is a Trojan that targets older Windows versions in order to open a back door in the system and allow the attacker to use the machine without the owner's consent.

ZeroAccess: this Trojan has the main purpose of assuring money to the attacker via pay-per-click advertisement. This tool is able to create a hidden and encrypted file system where it can save its members in total freedom, as well as all other additional malware that can download.

Zeus: has the main purpose of stealing information related to the bank accounts of the targets by means of techniques such as man-in the-browser, keystroke logging and form grabbing. The spread of the virus occurs mainly through drive-by downloads, initiated by mistake from the user, or phishing schemes. There is a server that acts as a control center from which the commands start.

6.5 Testbed and Scenario Setup

6.5.1 The network

The network is emulated by means of mininet [126], an open source program which allows recreating a realistic virtual network inside a pc. More precisely,

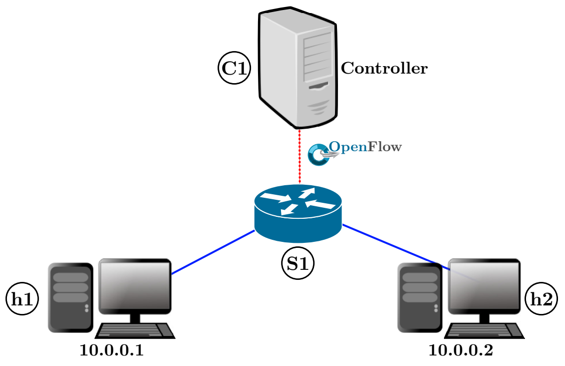


Figure 6.5: Network topology

it is a *network emulation orchestration system* that runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network.

Mininet supports the SDN paradigm by implementing its main components: the switches and controller. Regarding switches, mininet leverages *Open v-Switch* [37], an open source, multilayer virtual switch, explicitly designed to enable network automation through programmatic extensions, while still supporting standard management interfaces and protocols.

Concerning the controller, mininet implements the basic OpenFlow reference controller by default but it is possible to specify which controller to use at launch time. In our work we set as SDN controller the instance of Ryu that contains the `stats_manager` app running on the same machine that emulates the network.

The chosen network topology to perform the simulations is sketched in Figure 6.5. It consists of two hosts ($h1$ and $h2$) connected to a single SDN switch ($S1$) that communicate with the SDN controller ($C1$) through a dedicated channel.

6.5.2 Traffic emulation

In order to emulate a real scenario, we merged the flows produced by the malwares described in Section 6.4 with normal, malware-free traffic. We have captured normal flows by setting a network switch in our laboratory in port monitoring mode and sniffing all the traffic coming to the switch. Before starting the actual transmission, some manipulations on the original captured flows are performed with the help of the Tcpreplay [127] and Wireshark [128] command line tools:

- IP addresses was rewritten in such a way that all the packets appear to be exchanged between only two IP addresses: 10.0.0.1 and 10.0.0.2.
- In order to keep the simulation time reasonable, we cut the longest traces in order to have the same duration for normal and malware traffic. The duration of our experiments was approximately 2 hours.
- We divided the 12 malware's captures in 2 groups of 6 and added one malware-free capture to each group. In particular, **group 1** contained Asprox, Cutwail, Darkness, Madness, Purplehaze, and Zeus; while **group 2** contained Alienspy, Kuluoz, Neris, Ramnit, Tbot, and Zeroaccess.
- All the captured flows of both groups were temporally shifted in order to begin at the same time instant. Finally, they were merged together in two final pcap files to be used in the emulations.

6.5.3 The Classifier

Ensemble learning algorithms (e.g., random forest, bagging and boosting) have received an increasing interest because they are more accurate and robust to noise and outliers than single classifiers [129]. The philosophy behind ensembles classifiers is that a set of classifiers performs better than an individual classifier. [88] introduced a new and promising classifier in 2001 called Random Forest that consists of a collection of tree-structured classifiers, each one initialized with an independent identically distributed random vector $\boldsymbol{x}$. Random Forest presents many advantages: it runs efficiently on large databases, it is able to handle thousands of input variables without variable deletion, it is computationally lighter than other tree ensemble methods. Moreover, Random Forest estimates which variables are more important in the classification.

We performed a classifier performance evaluation in the previous chapter and Random Forest provided the best performance, for this reason, it has been also used also in the following tests.

When a sample is given as input to the classifier every tree independently decides the class of the samples $\hat{C}_b(\boldsymbol{x})$ and casts a unit vote for it. The most voted class is the final output of the classifier $\hat{C}^B_{rf} = mojorityvote\{\hat{C}_b(\boldsymbol{x})\}^B_1$.

The implementation of the classifier in python language is taken from the scikit-learn library [130], whose APIs are inserted into Ryu's code through a wrapper class we wrote for this purpose.

6.6 Performance evaluation and comments

Table 6.2 presents the overall results related to the two groups of malware described in Section 6.4. When group 2 is tested, group 1 is used in the training phase and viceversa, as shown in Table 6.2. The indicated values refer to the percentages of the parameters True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN), and Accuracy (ACC, computed as $\frac{TP+TN}{TF}$, where TF is the total number of flows).

Sim	Train	Test	TP	TN	FP	FN	ACC
1	group 1	group 2	0.845	0.987	0.013	0.155	0.887
2	group 2	group 1	0.978	0.964	0.036	0.022	0.972

Table 6.2: Results of the Simulations

In this particular application, since the result of the flows' classification is available for each time window, we computed an overall value for the aforementioned quantities. For each of them, a weighted average was computed as follows:

$$TP = \frac{\sum_{t=1}^{T} TP_t}{\sum_{t=1}^{T} F_t^m}, \quad TN = \frac{\sum_{t=1}^{T} TN_t}{\sum_{t=1}^{T} F_t^n}$$
$$FP = \frac{\sum_{t=1}^{T} FP_t}{\sum_{t=1}^{T} F_t^n}, \quad FN = \frac{\sum_{t=1}^{T} FN_t}{\sum_{t=1}^{T} F_t^m}$$

$$ACC = \frac{\sum_{t=1}^{T} TP_t + TN_t}{\sum_{t=1}^{T} F_t^T}$$

where:

- t is a counter of the *time window* when the classification takes place.

- T is the last time instant of the simulation.
- TP_t is the number of TP at instant t.
- F_t^m is the number of malware flows at instant t.
- F_t^n is the number of normal flows at instant t.
- F_t^T is the total number flows at instant t.

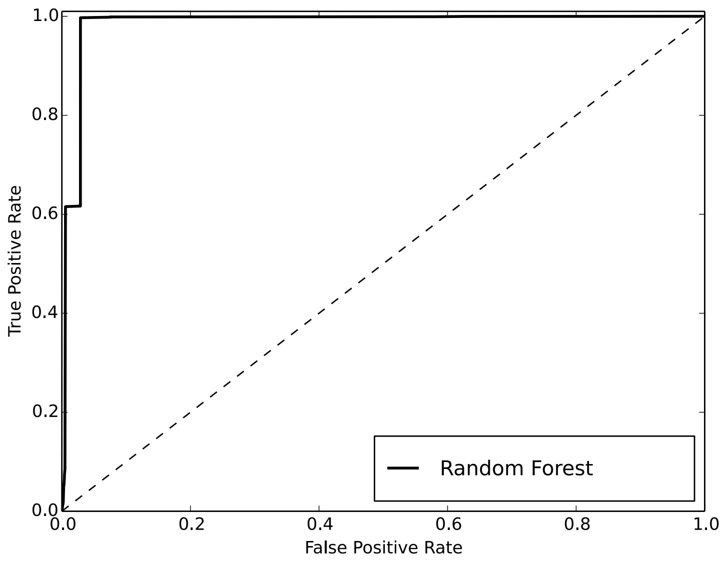


Figure 6.6: ROC Diagram of Simulation 1

Figures 6.6 and 6.7 show the Receiver Operating Characteristic (ROC) curves of the two simulations in Table 6.2. The dashed line is the "line of no discrimination" given by a random guess. The graphs show a performance very close to the ideal one (point $(0, 1)$) in both figures. It means that our designed system reaches a very high value of correct detections (TP) with a

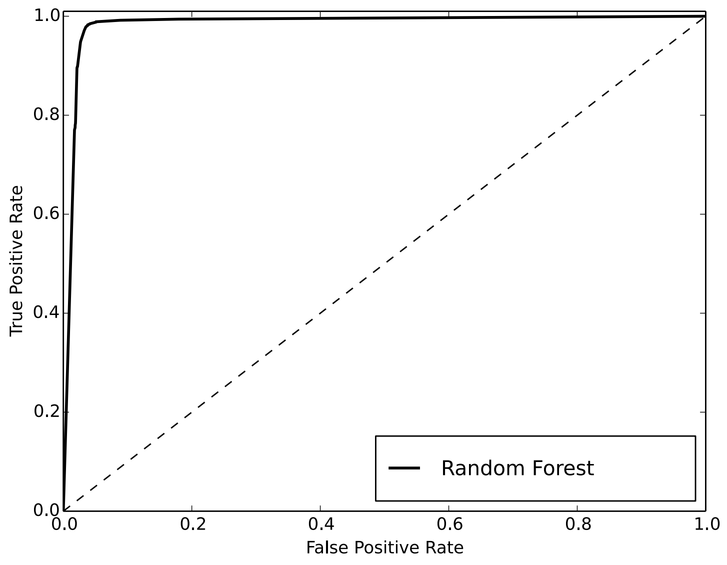


Figure 6.7: ROC Diagram of Simulation 2

very low probability of false alarms (FP). It is worth remarking that, in both cases, the classifier is trained with a set of malware flows which is different from the one employed in the testing phase, i.e. the malware used to test the classifier is not part of the dataset used to build the model for the Random Forest algorithm during the training phase.

The implementation of a combined malware detector IDS and SDN system allows simplifying the IDS design also improving the service offered by an SDN architecture. In this chapter we presented a possible implementation of an integrated SDN-IDS Ryu-based controller application devoted to detect the possible presence of malwares traversing the network. The results obtained through a simulation campaign have demonstrated the effectiveness and robustness of the proposed system, which has reached an accuracy level ranging from 88 and 97%.

This is a little step forward in the use of an SDN approach that can lead to innovative solutions to manage and secure networks.

Chapter 7

Conclusions

In the first part of this work we analyzed and explained the most important benefits and improvements that SDN architecture provided to traditional networking approaches. The main innovation brought by SDN is the decoupling of control plane and data plane. In SDN the two tasks are assigned to different entities: the controller, which is the part of the network dedicated to compute the forwarding state and the switch, which is the node devoted to packet forwarding based on the local forwarding state. Then, we described the concept behind Quality of Service and we illustrated the most common approaches used in traditional networks to assure specific performance requirements. The previous considerations allowed to understand the importance of Quality of Service in priority-based networks and its benefits in emerging applications. For this reason we analyzed the OpenFlow protocol and the basic tools it provides to support QoS solutions. Due to the lack of flexibility and power of the tools mentioned above, we decided to propose new approaches to manage priority flows inside a SDN network, exploiting the re-routing principle. The solutions introduced by our work are the Deadline algorithm, in which priority flows are assigned to the least loaded queue, and the Dedicated strategy, in which each queue of the SDN switch is devoted to a specific type of traffic. Moreover we developed several load balancing algorithms to the purpose of equalizing the traffic among the queues of a SDN node. Our cases of study show that the proposed QoS solutions allow to gain satisfying results when applied to the current OpenFlow environment. Future developments could consist in testing the network environment with a larger amount of traffic in order to test the scalability of our solutions. Furthermore, we plan to run our algorithms in other scenarios set with different queue configurations. We will investigate alternative QoS strategies that could lead to further improvements in performances. We also plan to develop an extension of our internal re-routing approach to the computation of alternative paths between source and destination, in order to leverage the network congestion.

After the discussion on the QoS support in a SDN network we focus our attention on the satellite environment. This implies to take into account the limitations and the possible advantages brought by this world. One of them is the fact that traditionally current operating satellite communication networks rely on ad-hoc hardware components and proprietary software solutions. This hinders the integration of satellite and terrestrial networks and also of different satellite networks. Fortunately, new solutions are arising to ease the integration and allow satellites to be part of an overall Internet composed of heterogeneous networks, according to the network evolution envisioned in the 5G environment. SDN is one of them but its employment in satellite networks is not straightforward. In this work we try to understand which is the current employments of this technology in the satellite environment surveying the outputs of some the main research projects and studies about the integration of satellite networks in the 5G environment, proposing a possible road-map for the introduction of the SDN and virtualization concepts in the next generation satellite - terrestrial networks and proposing a model to estimate the mean time required to complete the SDN control actions and to deliver the first packet of a new traffic flow able to give and idea of the delay introduced by the use of SDN in satellite communications.

Finally, we discussed about the network security and in particular we focused on the intrusion detection systems, which are systems aimed at analyzing and detecting security problems. The first part of the chapter is devoted to IDSs based on the taxonomy related to these systems. More precisely, we discussed on the differences between Misuse and Anomaly Detection based IDSs. Then we concentrate our attention on statistical fingerprint-based intrusion detection systems, which inspects packets in order to get a statistical characterization of the flows. This characterization represents their fingerprint and based on this our system makes decisions regarding the flow under analysis. More in detail, we extract a specific flow fingerprint able to tell us if the considered flow is affected by malware or not. For this purpose, we used different machine learning algorithms, which take as input the aforementioned fingerprint and release as output the category of the analyzed flow. After this we investigated if it was feasible to apply this technique also in the SDN environment. To this purpose, using the knowledge acquired during the entire research, we built an SDN controller application able to manage both the traffic traversing the network and, in the same time, identify if a considered flow is affected by malware or not. Both of these actions can be done by the used SDN controller Ryu without any OpenFlow modification, resulting 100% compliant with the current specifications.

This research is not intended finished but, on the contrary, there are a lot of things to do in the SDN field. A possible future work, which is in direct sequence with this thesis, can be understand what are the implication regarding the next generation (5G) networks, not only in the satellite environment, as done in this work, but also in the other fields treated inside this thesis. One possible example is the study of the implication of SDN in the support of quality of service in terrestrial 5G environment. An other interesting field of study in which SDN can play a major role, it is applying its concepts to industrial networks. Nowadays, industries and in particular the energy production environment are trying to open their network to the internet and the infrastructure remote control is becoming a must. In this transformation SDN can play a primary role due to its flexibility and easier management.

Bibliography

[1] *OpenFlow Switch Specification*, Open Networking Foundation, December 19, 2014, version 1.5.0.

[2] S. Shenker, “A Gentle Introduction to Software Defined Networks,” http://tce.technion.ac.il/files/2012/06/Scott-shenker.pdf, 2012, UC Berkeley, Technion Computer Engineering Center.

[3] T. Nadeau and K. Gray, *SDN - Software Defined Networks*, O’Reilly, Ed. O’Reilly Media Inc., 2013.

[4] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, “Software-defined networking (SDN): a survey,” *Security and Communication Networks*, vol. 9, no. 18, pp. 5803–5833, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1737

[5] Open Networking Fundation, “Software-Defined Networking: The New Norm for Networks,” April 2012.

[6] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKewon, and S. Shenker, “Nox: towards an operating system for networks,”

ACM SIGCOMM Computer Commun. Review, vol. 38, no. 3, p. 105–110, 2008.

[7] B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1617–1634, Third 2014.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[9] W. C. Hardy, *QoS Measurement and Evaluation of Telecommunications Quality of Service*, Wiley, Ed. John Wiley and Sons, Chichester, 2001.

[10] M. Marchese, *QoS Over Heterogeneous Networks*, Wiley, Ed. Wiley Publishing, 2007.

[11] R. Gellens, "The SYS and AUTH POP Response Codes," RFC 3206 (Proposed Standard), Internet Engineering Task Force, February 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3206.txt

[12] J. Gozdecki, A. Jajszczyk, and R. Stankiewicz, "Quality of service terminology in IP networks," *Communications Magazine, IEEE*, vol. 41, no. 3, pp. 153 – 159, March 2003.

[13] "ITU-T Recommendation. Support of IP based Services Using IP Transfer Capabilities. ITU-T Recommendation Y.1241," March 2001.

[14] "ITU-T Recommendation. Network Performance Objectives for IP-Based Services. ITU-T Recommendation Y.1541," February 2003.

[15] "ITU-T Recommendation. IP Packet Transfer and Availability Performance Parameters. ITU-T Recommendation Y.1540," November 2002.

[16] A. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall Professional Technical Reference, 2002.

[17] H. J. Chao and X. Guo, *Quality of Service Control in High-Speed Networks.* John Wiley and Sons, Chichester, England, 2002.

[18] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '07, 2007, pp. 1–12.

[19] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[20] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better Never Than Late: Meeting Deadlines in Datacenter Networks," in

Proceedings of the ACM SIGCOMM 2011 Conference, ser. SIGCOMM '11, 2011, pp. 50–61.

[21] D. Erickson, "The Beacon Openflow Controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13, 2013, pp. 13–18.

[22] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and S. G, "RSVP-TE: Extensions to RSVP for LSP tunnels," RFC 3209, 2001.

[23] D. Applegate and M. Thorup, "Load optimal MPLS routing with N + M labels," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 1, March 2003, pp. 555–565.

[24] N. Handigol, S. Seetharaman, M. Flajslik, R. Johari, and N. McKeown, "Aster*x: Load-Balancing as a Network Primitive," in *Plenary Demo, 9th GENI Engineering Conference*, ser. 9th GENI, November 2010.

[25] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based Server Load Balancing Gone Wild," in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, ser. Hot-ICE'11, 2011, pp. 12–12.

[26] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving High Utilization with Software-driven WAN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, 2013, pp. 15–26.

[27] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 2211–2219.

[28] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid Isolation: A Slice Abstraction for Software-defined Networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12, 2012, pp. 79–84.

[29] *OpenFlow Management and Configuration Protocol*, Open Networking Foundation, 2014, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf.

[30] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting Parallelism to Scale Software Routers," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 15–28. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629578

[31] T. Meyer, D. Raumer, F. Wohlfart, B. Wolfinger, and G. Carle, *Validated Model-Based Performance Prediction of Multi-Core Software Routers.* Praxis der Informationsverarbeitung und Kommunikation (PIK), 2014, ch. vol. 2, pp. 1–12.

[32] T. Meyer, D. Raumer, F. Wohlfart, B. E. Wolfinger, and G. Carle, "Low latency packet processing in software routers," in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2014)*, July 2014, pp. 556–563.

[33] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007. [Online]. Available: http://doi.acm.org/10.1145/1323293.1294281

[34] T. Hoff. (2009, Nov.) 10 eBay Secrets for Planet Wide Scaling. http://highscalability.com/blog/2009/11/17/10-ebay-secrets-for-planet-wide-scaling.html.

[35] W. Vogels. (2009, Apr.) Performance and Scalability. http://www.allthingsdistributed.com/2006/04/performance_and_scalability.html.

[36] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible Network Experiments Using Container-based Emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 253–264. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413206

[37] "Open vSwitch," http://openvswitch.org/, 2014.

[38] C. G. Cassandras, Y. Wardi, B. Melamed, G. Sun, and C. G. Panayiotou, "Perturbation analysis for online control and

optimization of stochastic fluid models," *IEEE Transactions on Automatic Control*, vol. 47, no. 8, pp. 1234–1248, Aug 2002.

[39] M. Cello, M. Marchese, and M. Mongelli, "On the qos estimation in an openflow network: The packet loss case," *IEEE Communications Letters*, vol. 20, no. 3, pp. 554–557, March 2016.

[40] R. E. Korf, "Multi-way Number Partitioning," in *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, ser. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 538–543. [Online]. Available: http://dl.acm.org/citation.cfm?id=1661445.1661531

[41] J. Ordóñez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Muñoz, J. Lorca, and J. Folgueira, "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, May 2017.

[42] *3GPP TS 23.501:"Technical Specification Group Services and Systems Aspects; System Architecture for the 5G system; Stage 2"*.

[43] Z. Ma, Z. Zhang, Z. Ding, P. Fan, and H. Li, "Key techniques for 5G wireless communications: network architecture, physical layer, and MAC layer perspectives," *Science China Information Sciences*, vol. 58, no. 4, pp. 1–20, Apr 2015.

[44] L. Bertaux, S. Medjiah, P. Berthou, S. Abdellatif, A. Hakiri, P. Gelard, F. Planchou, and M. Bruyere, "Software Defined

Networking and Virtualization for Broadband Satellite Networks," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 54–60, 2015.

[45] M. Sheng, Y. Wang, J. Li, R. Liu, D. Zhou, and L. He, "Toward a flexible and reconfigurable broadband satellite network: Resource management architecture and strategies," *IEEE Wireless Communications*, vol. 24, no. 4, pp. 127–133, 2017.

[46] J. Liu, Y. Shi, L. Zhao, Y. Cao, W. Sun, and N. Kato, "Joint placement of controllers and gateways in SDN-enabled 5G-satellite integrated network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 2, pp. 221–232, Feb 2018.

[47] T. Li, H. Zhou, H. Luo, and S. Yu, "Service: A software defined framework for integrated space-terrestrial satellite communication," *IEEE Transactions on Mobile Computing*, vol. 17, no. 3, pp. 703–716, 2017.

[48] Z. Georgios, P. Georgia, N. José, B. Jorge, M. Isaac, T. Christos, M. Sina, S. S. Krishna, A. Maha, and C. Symeon, "SANSA-hybrid terrestrial-satellite backhaul network: scenarios, use cases, KPIs, architecture, network and physical layer techniques," *International Journal of Satellite Communications and Networking*, vol. 35, no. 5, pp. 379–405, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sat.1207

[49] T. Ahmed, E. Dubois, J.-B. Dupé, R. Ferrús, P. Gélard, and N. Kuhn, "Software-defined satellite cloud RAN," *International*

Journal of Satellite Communications and Networking, vol. 36, no. 1, pp. 108–133, 2018.

[50] R. Ferrús, H. Koumaras, O. Sallent, G. Agapiou, T. Rasheed, M.-A. Kourtis, C. Boustie, P. Gélard, and T. Ahmed, "SDN/NFV-enabled satellite communications networks: Opportunities, scenarios and challenges," *Physical Communication*, vol. 18, pp. 95–112, 2016.

[51] G. Gardikis, H. Koumaras, C. Sakkas, and V. Koumaras, "Towards SDN/NFV-enabled satellite networks," *Telecommunication Systems*, vol. 66, no. 4, pp. 615–628, 2017.

[52] B. Yang, Y. Wu, X. Chu, and G. Song, "Seamless handover in software-defined satellite networking," *IEEE Communications Letters*, vol. 20, no. 9, pp. 1768–1771, 2016.

[53] T. Ahmed, R. Ferrús, R. Fedrizzi, O. Sallent, N. Kuhn, E. Dubois, and P. Gélard, "Satellite gateway diversity in SDN/NFV-enabled satellite ground segment systems," in *Communications Workshops (ICC Workshops), 2017 IEEE International Conference on*. IEEE, 2017, pp. 882–887.

[54] M. Mongelli, T. De Cola, M. Cello, M. Marchese, and F. Davoli, "Feeder-link outage prediction algorithms for SDN-based high-throughput satellite systems," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[55] T. Rossi, M. De Sanctis, E. Cianca, C. Fragale, M. Ruggieri, and H. Fenech, "Future space-based communications infrastructures based

on high throughput satellites and software defined networking," *IEEE International Symposium on Systems Engineering (ISSE)*, pp. 332–337, 2015.

[56] Y. Zhang and Y. Wang, "SDN based ICN architecture for the future integration network," *16th International Symposium on Communications and Information Technologies (ISCIT)*, pp. 474–478, 2016.

[57] J. Bao, B. Zhao, W. Yu, Z. Feng, C. Wu, and Z. Gong, "Opensan: a software-defined satellite network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 347–348, 2014.

[58] Z. Tang, B. Zhao, W. Yu, Z. Feng, and C. Wu, "Software defined satellite networks: Benefits and challenges," *IEEE Computing, Communications and IT Applications Conference (ComComAp)*, pp. 127–132, 2014.

[59] T. Li, H. Zhou, H. Luo, W. Quan, and S. Yu, "Modeling software defined satellite networks using queueing theory," *IEEE International Conference on Communications (ICC)*, pp. 1–6, 2017.

[60] J. Feng, L. Jiang, Y. Shen, W. Ma, and M. Yin, "A scheme for software defined ors satellite networking," *IEEE Fourth International Conference on Big Data and Cloud Computing (BdCloud)*, pp. 716–721, 2014.

[61] A. Ferreira, J. Galtier, and P. Penna, “Topological design, routing and handover in satellite networks,” *Handbook of wireless networks and mobile computing*, vol. 473, p. 493, 2002.

[62] F. Long, *Satellite Network Constellation Design.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 21–40. [Online]. Available: https://doi.org/10.1007/978-3-642-54353-1_2

[63] S. Cakaj, B. Kamo, A. Lala, and A. Rakipi, “The coverage analysis for low earth orbiting satellites at low elevation,” *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 5, no. 6, 2014.

[64] R. L. Douglas, *Satellite communications technology.* Englewood Cliff, NJ, USA: Prentice Hall, 1988.

[65] Y. Ye, T. Li, Q. Jiang, and Y. Wang, “Cimds: adapting postprocessing techniques of associative classification for malware detection,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 40, no. 3, pp. 298–307, 2010.

[66] V. G. Cerf, “Defense against the dark arts,” *Internet Computing, IEEE*, vol. 16, no. 1, pp. 96–96, 2012.

[67] H. Kim, K. G. Shin, and P. Pillai, “Modelz: monitoring, detection, and analysis of energy-greedy anomalies in mobile handsets,” *Mobile Computing, IEEE Transactions on*, vol. 10, no. 7, pp. 968–981, 2011.

[68] M. Chandramohan and H. B. K. Tan, “Detection of mobile malware in the wild,” *Computer*, vol. 45, no. 9, pp. 0065–71, 2012.

[69] M. La Polla, F. Martinelli, and D. Sgandurra, “A survey on security for mobile devices,” *Communications surveys & tutorials, IEEE*, vol. 15, no. 1, pp. 446–471, 2013.

[70] E. Fernandes, B. Crispo, and M. Conti, “Fm 99.9, radio virus: Exploiting fm radio broadcasts for malware deployment,” *Information Forensics and Security, IEEE Transactions on*, vol. 8, no. 6, pp. 1027–1037, 2013.

[71] S. Gianvecchio, M. Xie, Z. Wu, and H. Wang, “Humans and bots in internet chat: measurement, analysis, and automated classification,” *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 5, pp. 1557–1571, 2011.

[72] S. Y. Yerima, S. Sezer, and G. McWilliams, “Analysis of bayesian classification-based approaches for android malware detection,” *Information Security, IET*, vol. 8, no. 1, pp. 25–36, 2014.

[73] W. Peng, F. Li, X. Zou, and J. Wu, “Behavioral malware detection in delay tolerant networks,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 1, pp. 53–63, 2014.

[74] H. Nakayama, S. Kurosawa, A. Jamalipour, Y. Nemoto, and N. Kato, “A dynamic anomaly detection scheme for aodv-based mobile ad hoc networks,” *Vehicular Technology, IEEE Transactions on*, vol. 58, no. 5, pp. 2471–2481, 2009.

[75] F. Sabahi and A. Movaghar, "Intrusion detection: A survey," in *Systems and Networks Communications, 2008. ICSNC'08. 3rd International Conference on.* IEEE, 2008, pp. 23–26.

[76] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222–232, Feb 1987.

[77] K. Fukunaga, *Introduction to Statistical Pattern Recognition (2Nd Ed.).* San Diego, CA, USA: Academic Press Professional, Inc., 1990.

[78] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2Nd Edition).* Wiley-Interscience, 2000.

[79] A. McCallum, K. Nigam *et al.*, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752. Citeseer, 1998, pp. 41–48.

[80] P. Langley, W. Iba, and K. Thompson, "An analysis of bayesian classifiers," in *Aaai*, vol. 90, 1992, pp. 223–228.

[81] M. A. Hall and E. Frank, "Combining naive bayes and decision tables." in *FLAIRS Conference*, vol. 2118, 2008, pp. 318–319.

[82] B. R. Gaines and P. Compton, "Induction of ripple-down rules applied to modeling large databases," *Journal of Intelligent Information Systems*, vol. 5, no. 3, pp. 211–228, 1995.

[83] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods - Support*

Vector Learning. MIT Press, January 1998. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=68391

[84] J. R. Quinlan, *C4.5: Programs for Machine Learning.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[85] W. W. Cohen, "Fast effective rule induction," in *Proceedings of the twelfth international conference on machine learning*, 1995, pp. 115–123.

[86] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," in *ICML*, vol. 98, 1998, pp. 144–151.

[87] E. Frank, "Fully supervised training of gaussian radial basis function networks in weka," *Department of Computer Science, University of Waikato, Tech. Rep*, vol. 4, p. 14, 2014.

[88] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1010933404324

[89] J. J. Blount, D. R. Tauritz, and S. A. Mulder, "Adaptive rule-based malware detection employing learning classifier systems: a proof of concept," in *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual.* IEEE, 2011, pp. 110–115.

[90] W. Zhuang, Y. Ye, Y. Chen, and T. Li, "Ensemble clustering for internet security applications," *Systems, Man, and Cybernetics, Part*

C: Applications and Reviews, IEEE Transactions on, vol. 42, no. 6, pp. 1784–1796, 2012.

[91] Z. Shan and X. Wang, "Growing grapes in your computer to defend against malware," *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 2, pp. 196–207, 2014.

[92] P. O'Kane, S. Sezer, K. McLaughlin, and E. G. Im, "Svm training phase reduction using dataset feature filtering for malware detection," *Information Forensics and Security, IEEE Transactions on*, vol. 8, no. 3, pp. 500–509, 2013.

[93] I. Santos, F. Brezo, B. Sanz, C. Laorden, and P. G. Bringas, "Using opcode sequences in single-class learning to detect unknown malware," *Information Security, IET*, vol. 5, no. 4, pp. 220–227, 2011.

[94] S. Cesare, Y. Xiang, and W. Zhou, "Malwise - an effective and efficient classification system for packed and polymorphic malware," *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1193–1206, June 2013.

[95] A. Alhomoud, R. Munir, J. P. Disso, I. Awan, and A. Al-Dhelaan, "Performance evaluation study of intrusion detection systems," *Procedia Computer Science*, vol. 5, pp. 173 – 180, 2011, the 2nd International Conference on Ambient Systems, Networks and Technologies (ANT-2011) / The 8th International Conference on Mobile Web Information Systems (MobiWIS 2011). [Online].

Available:
http://www.sciencedirect.com/science/article/pii/S1877050911003498

[96] S. K. Cha, I. Moraru, J. Jang, J. Truelove, D. Brumley, and D. G. Andersen, “Splitscreen: Enabling efficient, distributed malware detection,” *Communications and Networks, Journal of*, vol. 13, no. 2, pp. 187–200, 2011.

[97] M. A. Aydın, A. H. Zaim, and K. G. Ceylan, “A hybrid intrusion detection system design for computer network security,” *Computers & Electrical Engineering*, vol. 35, no. 3, pp. 517–526, 2009.

[98] H. Om and A. Kundu, “A hybrid system for reducing the false alarm rate of anomaly intrusion detection system,” in *Recent Advances in Information Technology (RAIT), 2012 1st International Conference on*. IEEE, 2012, pp. 131–136.

[99] R. R. Karthick, V. P. Hattiwale, and B. Ravindran, “Adaptive network intrusion detection system using a hybrid approach,” in *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*. IEEE, 2012, pp. 1–7.

[100] R. M. Elbasiony, E. A. Sallam, T. E. Eltobely, and M. M. Fahmy, “A hybrid network intrusion detection framework based on random forests and weighted k-means,” *Ain Shams Engineering Journal*, vol. 4, no. 4, pp. 753–762, 2013.

[101] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli, “Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting,” *Computer Networks*, vol. 53, no. 1, pp. 81–97, 2009.

[102] M. Aiello, M. Mongelli, and G. Papaleo, “Dns tunneling detection through statistical fingerprints of protocol messages and machine learning,” *International Journal of Communication Systems*, vol. 28, no. 14, pp. 1987–2002, 2015.

[103] H. Nakayama, A. Jamalipour, and N. Kato, “Network-based traitor-tracing technique using traffic pattern,” *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 2, pp. 300–313, June 2010.

[104] M. N. Mohammad, N. Sulaiman, and O. A. Muhsin, “A novel intrusion detection system by using intelligent data mining in weka environment,” *Procedia Computer Science*, vol. 3, pp. 1237–1242, 2011.

[105] M. Zamani and M. Movahedi, “Machine learning techniques for intrusion detection,” *arXiv preprint arXiv:1312.2177*, 2013.

[106] G. V. Nadiammai and M. Hemalatha, “Perspective analysis of machine learning algorithms for detecting network intrusions,” in *Computing Communication Networking Technologies (ICCCNT), 2012 Third International Conference on*, July 2012, pp. 1–7.

[107] “Kdd cup 1999 data,” http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[108] S. Saravanan, S. Vijay Bhanu, and R. Chandrasekaran, "Study on classification algorithms for network intrusion systems," *Journal of Communication and Computer*, vol. 9, no. 11, pp. 1242–1246, 2012.

[109] A. C. Enache and V. V. Patriciu, "Intrusions detection based on support vector machine optimized with swarm intelligence," in *Applied Computational Intelligence and Informatics (SACI), 2014 IEEE 9th International Symposium on*, May 2014, pp. 153–158.

[110] H. Nakayama, A. Jamalipour, and N. Kato, "Network-based traitor-tracing technique using traffic pattern," *Information Forensics and Security, IEEE Transactions on*, vol. 5, no. 2, pp. 300–313, 2010.

[111] J. Postel, "Rfc 790—assigned numbers," 1981.

[112] J. Reynolds, "Assigned numbers: Rfc 1700 is replaced by an on-line database," 2002.

[113] T.-F. Yen, X. Huang, F. Monrose, and M. K. Reiter, "Browser fingerprinting from coarse traffic summaries: Techniques and implications," in *Proceedings of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 157–175.

[114] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, pp. 5–16, Jan. 2007. [Online]. Available: http://doi.acm.org/10.1145/1198255.1198257

[115] www.mediafire.com/?a49l965nlayad.

[116] "Weka – data mining machine learning software," http://www.cs.waikato.ac.nz/ml/weka/.

[117] R. K. Dash, "Selection of the best classifier from different datasets using weka," in *International Journal of Engineering Research and Technology*, vol. 2, no. 3 (March-2013). ESRSA Publications, 2013.

[118] H. A. Nguyen and D. Choi, "Application of data mining to network intrusion detection: classifier selection model," in *Challenges for Next Generation Network Operations and Service Management.* Springer, 2008, pp. 399–408.

[119] W. Stallings, "Software-defined networks and openflow," *The internet protocol Journal*, vol. 16, no. 1, pp. 2–14, 2013.

[120] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[121] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies.* ACM, 2013, pp. 25–30.

[122] R. Skowyra, S. Bahargam, and A. Bestavros, "Software-defined ids for securing embedded mobile devices," in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE.* IEEE, 2013, pp. 1–7.

[123] P. Wang, K.-M. Chao, H.-C. Lin, W.-H. Lin, and C.-C. Lo, "An efficient flow control approach for sdn-based network threat detection and migration using support vector machine," in *e-Business Engineering (ICEBE), 2016 IEEE 13th International Conference on*. IEEE, 2016, pp. 56–63.

[124] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*. IEEE, 2010, pp. 408–415.

[125] Ryu, "Framework community: Ryu sdn controller," https://osrg.github.io/ryu/, 2016.

[126] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *9th ACM Workshop on Hot Topics in Networks*, October 2010.

[127] A. Turner and M. Bing, "Tcpreplay," https://tcpreplay.appneta.com/, 2011.

[128] G. Combs *et al.*, "Wireshark," *Web page: http://www. wireshark. org/last modified*, pp. 12–02, 2007.

[129] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine learning*, vol. 40, no. 2, pp. 139–157, 2000.

[130] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg,

J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.