UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

UNIVERSITAT POLITÈCNICA DE CATALUNYA

DOCTORAL THESIS

Knowledge-Defined Networking: A Machine Learning based approach for network and traffic modeling

---

Author:
Albert MESTRES

Supervisors:
Dr. Albert CABELLOS
Dr. Eduard ALARCÓN

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

in the

Broadband Communications Research Group
Department or Computer Architecture

September 28, 2017

Declaration of Authorship

I, Albert MESTRES, declare that this thesis titled, "Knowledge-Defined Networking: A Machine Learning based approach for network and traffic modeling" and the work presented in it are my own. I confirm that:

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

Signed:

Date:

"Intelligence is the ability to avoid doing work,
yet getting the work done "

Linus Torvalds

Universitat Politècnica de Catalunya

Abstract

Department or Computer Architecture

Doctor of Philosophy in Computer Architecture

Knowledge-Defined Networking:
A Machine Learning based approach for network and traffic modeling

by Albert MESTRES

The research community has considered in the past the application of Machine Learning (ML) techniques to control and operate networks. A notable example is the Knowledge Plane proposed by D.Clark et al. However, such techniques have not been extensively prototyped or deployed in the field yet. In this thesis, we explore the reasons for the lack of adoption and posit that the rise of two recent paradigms: Software-Defined Networking (SDN) and Network Analytics (NA), will facilitate the adoption of ML techniques in the context of network operation and control. We describe a new paradigm that accommodates and exploits SDN, NA and ML, and provide use-cases that illustrate its applicability and benefits. We also present some relevant use-cases, in which ML tools can be useful. We refer to this new paradigm as Knowledge-Defined Networking (KDN).

In this context, ML can be used as a network modeling technique to build models that estimate the network performance. Network modeling is a central technique to many networking functions, for instance in the field of optimization. One of the objective of this thesis is to provide an answer to the following question: *Can neural networks accurately model the performance of a computer network as a function of the input traffic?*. In this thesis, we focus mainly on modeling the average delay, but also on estimating the jitter and the packets lost. For this, we assume the network as a black-box that has as input a traffic matrix and as output the desired performance matrix. Then we train different regressors, including deep neural networks, and evaluate its accuracy under different fundamental network characteristics: topology, size, traffic intensity and routing. Moreover, we also study the impact of having multiple traffic flows between each pair of nodes.

We also explore the use of ML techniques in other network related fields. One relevant application is traffic forecasting. Accurate forecasting enables scaling up or down the resources to efficiently accommodate the load of traffic. Such models are typically based on traditional time series ARMA or ARIMA models. We propose

a new methodology that results from the combination of an ARIMA model with an ANN. The Neural Network greatly improves the ARIMA estimation by modeling complex and nonlinear dependencies, particularly for outliers. In order to train the Neural Network and to improve the outliers estimation, we use external information: weather, events, holidays, etc. The main hypothesis is that network traffic depends on the behavior of the end-users, which in turn depend on external factors. We evaluate the accuracy of our methodology using real-world data from an egress Internet link of a campus network. The analysis shows that the model works remarkably well, outperforming standard ARIMA models.

Another relevant application is in the Network Function Virtualization (NFV). The NFV paradigm makes networks more flexible by using Virtual Network Functions (VNF) instead of dedicated hardware. The main advantage is the flexibility offered by these virtual elements. However, the use of virtual nodes increases the difficulty of modeling such networks. This problem may be addressed by the use of ML techniques, to model or to control such networks. As a first step, we focus on the modeling of the performance of single VNFs as a function of the input traffic. In this thesis, we demonstrate that the CPU consumption of a VNF can be estimated only as a function of the input traffic characteristics.

Acknowledgements

I would like to thank everybody who has made possible this thesis. First of all, I would like to thank my project advisors, Dr. Albert Cabellos and Dr. Eduard Alarcón for their help, patience and guidance they provided me during these more than four years of the PHD.

I would also like to acknowledge all members of the research group and the ones we have shared the space in our research laboratory, in which I enjoyed an amazing atmosphere. The members of the nanocomunications group deserve a special mention, with whom we shared many experience, specially during the firsts years of this thesis: Ignacio Llatser, Sergi Abadal and Raül Gómez. I would also like to mention Patricia Sampedro and Giorgio Stampa, with whom we shared the research topic during the last year of the thesis.

I would like to acknowledge the rest of the research group faculty and staff for their valuable comments and help. I would also like to thank the economical support from the FI scholarship from AGAUR (Agency for Management of University and Research Grants).

I do not want to forget all the friends from Barcelona, and the rest of the world, I have made along my life. And last but not least, I will be eternally thankful to my family for their love, support and understanding throughout my studies.

Contents

Declaration of Authorship iii

Abstract vii

Acknowledgements ix

1 Knowledge Defined Networking 1
- 1.1 Introduction . . . 1
- 1.2 A Knowledge Plane for SDN Architectures . . . 2
- 1.3 Knowledge-Defined Networking . . . 4
- 1.4 Use-cases . . . 7
  - 1.4.1 Routing in an Overlay Network . . . 8
  - 1.4.2 Resource Management in an NFV scenario . . . 9
  - 1.4.3 Knowledge extraction from network logs . . . 10
  - 1.4.4 5G mobile communications networks . . . 10
  - 1.4.5 Network Planning . . . 11
- 1.5 Challenges . . . 11
- 1.6 Thesis scope . . . 13
  - 1.6.1 Network performance modeling . . . 13
  - 1.6.2 Traffic characterization and forecasting . . . 14
  - 1.6.3 Complex network elements characterization . . . 14
  - 1.6.4 Thesis outline . . . 14

2 State of the art 17
- 2.1 Machine Learning . . . 17
  - 2.1.1 Machine Learning Approaches . . . 18
  - 2.1.2 Machine Learning techniques . . . 18
    - Artificial Neural Networks . . . 19
- 2.2 Networking: Novel networking paradigms and network modeling approaches . . . 21
  - 2.2.1 Software-Defined Networking and Network Analytics . . . 21
  - 2.2.2 Network Function Virtualization . . . 22
  - 2.2.3 Network Modeling . . . 22
  - 2.2.4 Markov chains and queuing theory . . . 23
  - 2.2.5 Network Calculus . . . 23

2.2.6 Discrete-Event Simulations . . . 24
2.2.7 Our approach . . . 24
2.3 Traffic modeling and prediction . . . 24
2.3.1 Statistical models . . . 25
2.3.2 Trace-based simulations . . . 25
2.3.3 Traffic forecasting . . . 26
2.4 Machine Learning in the networking field . . . 26

3 A theoretical inspired approach **29**
3.1 Introduction . . . 29
3.2 Problem statement . . . 30
3.3 Methodology . . . 30
3.4 Machine learning techniques . . . 31
3.4.1 A M/M/1-Inspired regressor . . . 31
Delays in M/M/1 queuing networks . . . 32
Model definition . . . 32
3.4.2 ANN models . . . 33
3.5 Experiment 1: Comparison and Evaluation of the M/M/1 model . . . 34
3.5.1 Overview . . . 34
3.5.2 Results . . . 34
3.6 Experiment 2: Global vs local view modeling . . . 37
3.6.1 Overview . . . 37
3.6.2 Results . . . 38
3.7 Conclusions . . . 38

4 Network modeling using ANN **39**
4.1 Problem Statement . . . 40
4.2 State-of-the-art . . . 41
4.3 Methodology . . . 42
4.3.1 Overview . . . 42
4.3.2 Network Simulations . . . 43
4.3.3 Regressors . . . 44
4.4 Experimental Results . . . 45
4.4.1 Traffic Characteristics and Intensity . . . 45
4.4.2 Topologies and Network Size . . . 50
4.4.3 Routing . . . 52
4.5 Discussion . . . 53
4.6 Conclusions . . . 54

5 Improving the traffic forecasting with ML **55**
5.1 Introduction . . . 55
5.2 Methodology . . . 56
5.2.1 Overview . . . 56

5.2.2 Time Series . . . 57
Example of Outliers Detection (MA(1) to AR(∞)) . . . 58
5.2.3 Neural Networks . . . 59
5.2.4 Incorporating the Neural Network to the ARIMA model . . . 60
5.3 Experimental Evaluation . . . 60
5.3.1 Dataset . . . 60
5.3.2 Neural Network . . . 61
5.3.3 Results . . . 62
5.4 Discussion and Conclusions . . . 64

6 Complex network elements modeling **67**
6.1 Introduction . . . 67
6.2 Problem Statement . . . 68
6.3 Methodology . . . 69
6.3.1 Overview . . . 69
6.3.2 Traffic . . . 70
6.3.3 VNFs . . . 70
6.3.4 Set Up . . . 72
6.3.5 Machine Learning training . . . 72
6.4 Experimental results . . . 73
6.4.1 Results and discussion . . . 73
6.5 Summary and Concluding Remarks . . . 75

7 Conclusions and future work **77**
7.1 Conclusions . . . 77
7.2 Future work . . . 78
7.2.1 KDN challenges . . . 78
7.2.2 Network and traffic modeling challenges . . . 79
7.2.3 Traffic forecasting challenges . . . 79
7.2.4 Complex network elements modeling challenges . . . 80
7.2.5 Topology and configuration challenges . . . 80

A Network Simulator **81**
A.1 Simulator requirements . . . 81
A.2 Design . . . 82
A.3 Use of the simulator . . . 83

Bibliography **85**

List of Figures

1.1 KDN planes . . . 3
1.2 KDN operational loop . . . 4
1.3 Overlay network with a hidden underlay . . . 8

2.1 KDN planes . . . 19

3.1 Evolution of the relative test error as a function of the training set size 35
3.2 Evolution of the relative test error as a function of the number of actively transmitting stations in the network . . . 36
3.3 Evolution of the relative test error as a function of the average level of saturation in the links (logarithmic scale) . . . 36
3.4 Prediction error (Mean Square Error) as a function of the size of the training set in an overlay-underlay scenario, both for global and local view. . . . 37

4.1 Graphical representation of the problem statement addressed in this chapter. . . . 40
4.2 Scheme of the methodology followed in this work . . . 43
4.3 Learning error (log scale) as a function of the network saturation for a binomial traffic . . . 45
4.4 Learning error (lineal scale) as a function of the network saturation for a binomial traffic in more saturated networks . . . 46
4.5 Learning error as a function of the averaging time . . . 47
4.6 Learning error (log scale) as a function of the network saturation for a Poisson traffic . . . 48
4.7 Learning error (log scale) as a function of the network saturation for a uniform traffic . . . 48
4.8 Learning error (log scale) as a function of the network saturation for a deterministic traffic . . . 49
4.9 Scale-free topology . . . 50
4.10 Star topology . . . 51
4.11 Ring topology . . . 51

5.1 Methodology scheme . . . 56
5.2 Time Series Decomposition (X axis: Days, Y axis: Bandwidth) . . . 57
5.3 Neural Network Structure with three levels . . . 60

5.4 Outliers impact on the series 61
5.5 Forecast of 7 days using the standard ARIMA model and the modified one. Top: Forecast with an AO outlier. Bottom: Forecast with a TC outlier. 63
5.6 Prediction Error Cumulative Density Function 65

6.1 Graphic representation of the VNF performance modeling addressed in this chapter 68
6.2 Scheme describing the configuration set up and the different VM used 70
6.3 Model built using two different features for two different VNF (only the relevant feature is shown) 73
6.4 Predicted value vs actual value in the three different configurations . . 74
6.5 Cumulative distribution function of the error as a function of the percentual error 75
6.6 Sorted training error of single features 76

List of Tables

1.1 KDN applications 6

3.1 Best cross-validated parameters 35

4.1 Learning error using different routing 52

6.1 Complete set of features used to describe the traffic 71

List of Abbreviations

AI	**A**rtificial **I**ntelligence
ANN	**A**rtificial **N**eural **N**etwork
ARIMA	**A**uto**R**egressive **I**ntegrated **M**oving **A**verage
ARMA	**A**uto**R**egressive **M**oving **A**verage
DC	**D**ata **C**enter
DL	**D**eep **L**earning
DNN	**D**eep **N**eural **N**etwork
KDN	**K**nowledge **D**efined **N**etworking
KP	**K**nowledge **P**lane
ML	**M**achine **L**earning
MSE	**M**ean **S**quared **E**rror
NA	**N**etwork **A**nalytics
NFV	**N**etwork **F**unction **V**irtualization
OVS	**O**pen **vS**witch
SDN	**S**oftware-**D**efined **N**etworking
VM	**V**irtual **M**achine
VNF	**V**irtual **N**etwork **F**unction

Chapter 1

Knowledge Defined Networking

1.1 Introduction

D. Clark et al. proposed "A Knowledge Plane for the Internet" [1], a new construct that relies on Machine Learning (ML) and cognitive techniques to operate the network. A Knowledge Plane (KP) would bring many advantages to networking, such as automation (recognize-act) and recommendation (recognize-explain-suggest), and it has the potential to represent a paradigm shift on the way we operate, optimize and troubleshoot data networks. However, at the time of this writing, we are yet to see the KP prototyped or deployed. Why?

One of the biggest challenges when applying ML for network operation and control is that networks are inherently distributed systems, where each node (i.e., switch, router) has only a partial view and control over the complete system. Learning from nodes that can only view and act over a small portion of the system is very complex, particularly if the end goal is to exercise control beyond the local domain. The emerging trend towards logical centralization of control will ease the complexity of learning in an inherently distributed environment. In particular, the Software-Defined Networking (SDN) paradigm [2] decouples control from the data plane and provides a logically centralized control plane, i.e., a logical single point in the network with knowledge of the whole.

In addition to the "softwarization" of the network, current network data plane elements, such as routers and switches, are equipped with improved computing and storage capabilities. This has enabled a new breed of network monitoring techniques, commonly referred to as network telemetry [3]. Such techniques provide real-time packet and flow-granularity information, as well as configuration and network state monitoring data, to a centralized Network Analytics (NA) platform. In this context, telemetry and analytics technologies provide a richer view of the network compared to what was possible with conventional network management approaches.

In this thesis, we advocate that the centralized control offered by SDN, combined with a rich centralized view of the network provided by network analytics, enable the deployment of the KP concept proposed in [1]. In this context, the KP can use various ML approaches, such as Deep Learning (DL) techniques, to gather knowledge about the network, and exploit that knowledge to control the network using logically centralized control capabilities provided by SDN. We refer to the paradigm resulting from combining SDN, telemetry, Network Analytics, and the Knowledge Plane as Knowledge-Defined Networking (KDN).

The following section describes the KDN paradigm and how it operates. Then, we describe a set of relevant use-cases that show the applicability of such paradigm to networking and the benefits associated with using ML. In addition, we analyze the open research challenges associated with the KDN paradigm. Finally, we define the scope of this thesis, which summarized is to demonstrate the feasibility of the KDN paradigm by modeling the end-to-end delay of a network using only traffic information using state-of-the-art techniques.

1.2 A Knowledge Plane for SDN Architectures

The concept of KDN introduced in this thesis restates the concept of Knowledge Plane (KP) as defined by D. Clark et al. [1] in the context of SDN architectures. The addition of a KP to the traditional three planes of the SDN paradigm results in what we call Knowledge-Defined Networking. Fig. 1.1 shows an overview of the KDN paradigm and its functional planes.

The *Data Plane* is responsible for storing, forwarding and processing data packets. In SDN networks, data plane elements are typically network devices composed of line-rate programmable forwarding hardware. They operate unaware of the rest of the network and rely on the other planes to populate their forwarding tables and update their configuration.

The *Control Plane* exchanges operational state in order to update the data plane matching and processing rules. In an SDN network, this role is assigned to the –logically centralized– SDN controller that programs SDN data plane forwarding elements via a southbound interface. While the data plane operates at packet time scales, the control plane is slower and typically operates at flow time scales.

The *Management Plane* ensures the correct operation and performance of the network in the long term. It defines the network topology and handles the provision and configuration of network devices. In SDN this is usually handled by the SDN controller as well. The management plane is also responsible for monitoring the network to provide critical network analytics. To this end, it collects telemetry information from the control and data plane while keeping a historical record of the

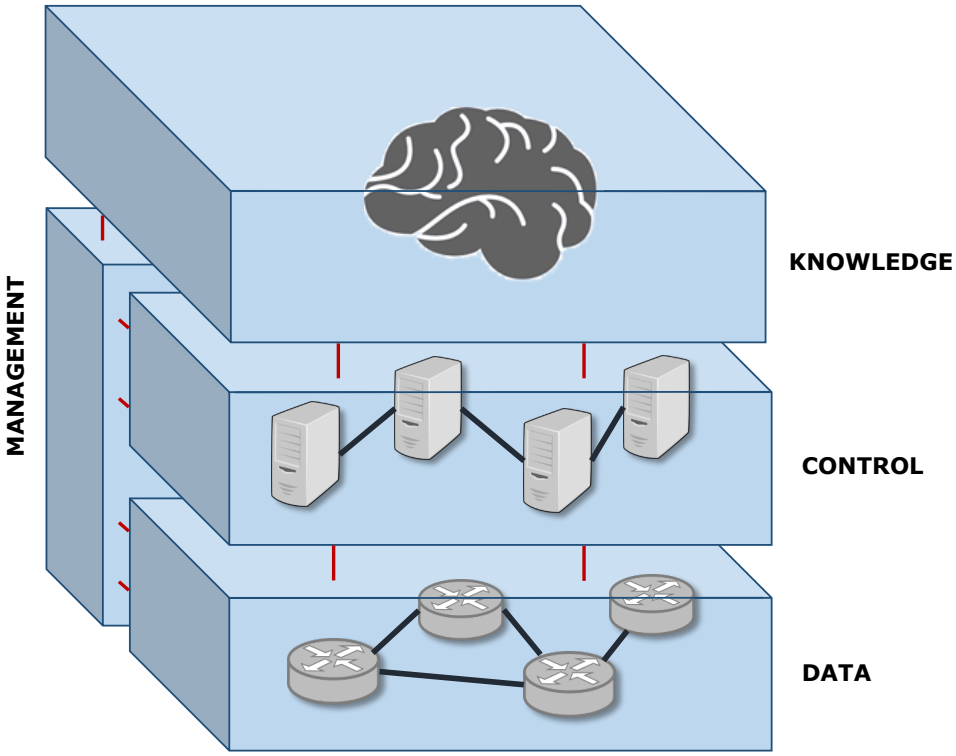

FIGURE 1.1: KDN planes

network state and events. The management plane is orthogonal to the control and data planes, and typically operates at larger time-scales.

The *Knowledge Plane*, as originally proposed by Clark, is redefined in this thesis under the terms of SDN as follows: *the heart of the knowledge plane is its ability to integrate behavioral models and reasoning processes oriented to decision making into an SDN network*. In the KDN paradigm, the KP takes advantage of the control and management planes to obtain a rich view and control over the network. It is responsible for learning the behavior of the network and, in some cases, automatically operate the network accordingly. Fundamentally, the KP processes the network analytics collected by the management plane, either preprocessed data or raw data, transforms them into knowledge via ML, and uses that knowledge to make decisions (either automatically or through human intervention). While parsing the information and learning from it is typically a slow off-line process, using such knowledge automatically can be done at a time-scales close to those of the control and management planes. However, the trend is towards on-line learning for applications such as those described in section 1.4.

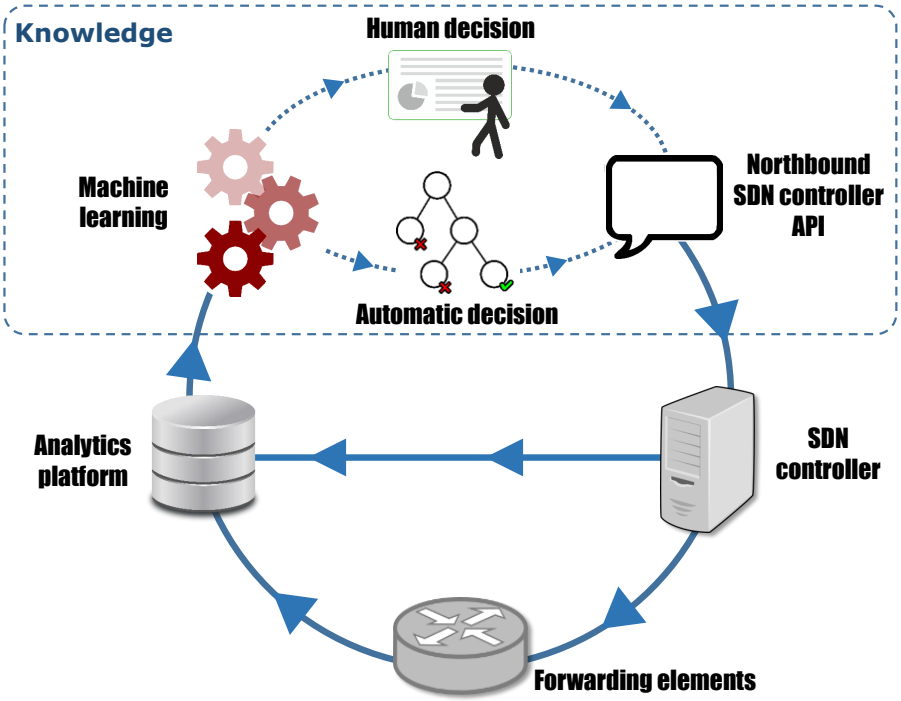

FIGURE 1.2: KDN operational loop

1.3 Knowledge-Defined Networking

The Knowledge-Defined Networking (KDN) paradigm operates by means of a control loop to provide automation, recommendation, optimization, validation and estimation. Conceptually, the KDN paradigm borrows many ideas from other areas, notably from black-box optimization [4], neural-networks in feedback control systems [5], reinforcement learning [6] and autonomic self-* architectures [6]. In addition, recent initiatives share the same vision stated in this thesis[1] [7–9] Fig. 2 shows the basic steps of the main KDN control. In what follows we describe these steps in detail.

Forwarding Elements & SDN Controller → Analytics Platform The *Analytics Platform* aims to gather enough information to offer a complete view of the network. To that end, it monitors the data plane elements in real time while they forward packets in order to access fine-grained traffic information. In addition, it queries the SDN controller to obtain control and management state. The analytics platform relies on protocols, such as NETCONF (RFC 6241), NetFlow (RFC 3954) and IPFIX (RFC 7011), to obtain configuration information, operational state and traffic data from the network. The most relevant data collected by the analytics platform is summarized below.

[1] Cognet project: http://www.cognet.5g-ppp.eu/

- Packet-level and flow-level data: This includes Deep Packet Inspection (DPI) information, flow granularity data and relevant traffic features.
- Network state: This includes the physical, topological and logical configuration of the network.
- Control & management state: This includes all the information included both in the SDN controller and management infrastructure, including policy, virtual topologies, application-related information, etc.
- Service-level telemetry: This is relevant to learn the behavior of the application or service, and its relation with the network performance, load and configuration.
- External information: This is relevant to model the impact of external events, such as activity on social networks (e.g., amount of people attending a sports event), weather forecasts, etc. on the network.

In order to effectively learn the network behavior, besides having a rich view of the network, it is critical to observe as many different situations as possible. As we discuss in Section 1.5, this includes different loads, configurations and services. To that end, the analytics platform keeps a historical record of the collected data.

Analytics Platform → Machine Learning ML algorithms (such as Deep Learning techniques) are the heart of the KP, which are able to learn from the network behavior. The current and historical data provided by the analytics platform are used to feed learning algorithms that learn from the network and generate knowledge (e.g., a model of the network). We consider three approaches: supervised learning, unsupervised learning and reinforcement learning.

In **supervised learning**, the KP learns a model that describes the behavior of the network, i.e., a function that relates relevant network variables to the operation of the network (e.g., the performance of the network as a function of the traffic load and network configuration). It requires labeled training data and feature engineering to represent network data.

Unsupervised learning is a data-driven knowledge discovery approach that can automatically infer a function that describes the structure of the analyzed data or can highlight correlations in the data that the network operator may be unaware of. As an example, the KP may be able to discover how the local weather affects the link's utilization.

In the **reinforcement learning** approach, a software agent aims to discover which actions lead to an optimal configuration. As an example the network administrator can set a target policy, for instance the delay of a set of flows, then the agent acts on the SDN controller by changing the configuration and for each action receives a reward,

TABLE 1.1: KDN applications

	Closed Loop	**Open Loop**
Supervised	Automation Optimization	Validation Estimation What-if analysis
Unsupervised	Improvement	Recommendation
Reinforcement	Automation Optimization	N/A

which increases as the in-place policy gets closer to the target policy. Ultimately, the agent will learn the set of configuration updates (actions) that result in such target policy. Recently, deep reinforcement learning techniques have provided important breakthroughs in the AI field that are being applied in many network-related fields (e.g., [10]).

Please note that learning can also happen offline and applied online. In this context knowledge can be learned offline training a neural network with datasets of the behavior of a large set of networks, then the resulting model can be applied online.

Machine Learning → Northbound controller API The KP eases the transition between telemetry data collected by the analytics platform and control specific actions. Traditionally, a network operator had to examine the metrics collected from network measurements and make a decision on how to act on the network. In KDN, this process is partially offloaded to the KP, which is able to make -or recommend- control decisions taking advantage of ML techniques.

Depending on whether the network operator is involved or not in the decision making process, there are two different sets of applications for the KP. We next describe these potential applications and summarize them in table 1.1.

- **Closed loop:** When using supervised or reinforcement learning, the network model obtained can be used first for *automation*, since the KP can make decisions automatically on behalf of the network operator. Second, it can be used for *optimization* of the existing network configuration, given that the learned network model can be explored through common optimization techniques to find (quasi)optimal configurations. In the case of unsupervised learning, the knowledge discovered can be used to automatically *improve* the network via the interface offered by the SDN controller. For instance the relation between traffic, routing, topology and the resulting delay can be modeled to then apply optimal routing configurations that minimize delay.
- **Open loop:** In this case the network operator is still in charge of making the decisions, however it can rely on the KP to ease this task. When using supervised learning, the model learned by ML can be used for *validation* (e.g., to

query the model before applying tentative changes to the system). The model can also be used as a tool for performance *estimation* and *what-if analysis*, since the operator can tune the variables considered in the model and obtain an assessment of the network performance. When using unsupervised learning, the correlations found in the explored data may serve to provide *recommendations* that the network operator can take into consideration when making decisions.

Northbound controller API → SDN controller The northbound controller API offers a common interface to, human, software-based network applications and policy makers to control the network elements. The API offered by the SDN controller can be either a traditional imperative language or a declarative one [11]. In the latter case, the users of the API express their intentions towards the network, which then are translated into specific control directives.

The KP can operate both on top of imperative or declarative languages as long as it is trained accordingly. However, and at the time of this writing, developing truly expressive and high-level declarative northbound APIs is an open research question. Such intent-based declarative languages provide automation and intelligence capabilities to the system. In this context, we advocate that the KP represents an opportunity to help on their development, rather than an additional level of intelligence. As a result, we envision the KP operating on top of imperative languages, while helping on the translation of the intentions stated by the policy makers into network directives.

SDN controller → Forwarding Elements The parsed control actions are pushed to the forwarding devices via the controller southbound protocols in order to program the data plane according to the decisions made at the KP.

1.4 Use-cases

This section presents a set of specific uses-cases that illustrate the potential applications of the KDN paradigm and the benefits a KP based on ML may bring to common networking problems. For two representative use-cases, we also provide early experimental results that show the technical feasibility of the proposed paradigm. All the datasets used in this thesis, as well as codes and relevant hyper-parameters, can be found at [12].

The KDN architecture we introduce in this thesis has a very wide range of use-cases in networking that take base on the use of ML techniques. In what follows we describe a set of specific use-cases that exemplify the applications of the KDN architecture.

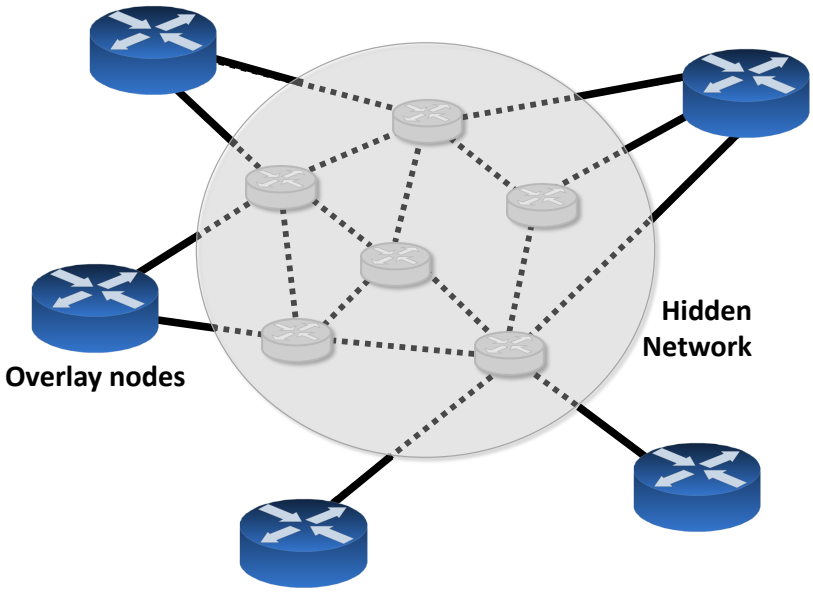

FIGURE 1.3: Overlay network with a hidden underlay

1.4.1 Routing in an Overlay Network

The main objective of this use-case is to show that it is possible to model the behavior of a network with the use of ML techniques. In particular, we present a simple proof-of-concept example in the context of overlay networks, where an Artificial Neural Network (ANN) is used to build a model of the delay of the (hidden) underlay network, which can later be used to improve routing in the overlay network.

Overlay networks have become a common solution for deployments where one network (overlay) has to be instantiated on top of another (underlay). This may be the case when a physically distributed system needs to behave as a whole while relying on a transit network, for instance a company with geo-distributed branches that connects them through the Internet. Another case is when a network has to send traffic through another for which it is not interoperable, for example when trying to send Ethernet frames over an IP-only network.

In such cases, an overlay network can be instantiated by means of deploying overlay-enabler nodes at the edge of the transit network and then tunneling overlay traffic using an encapsulation protocol (e.g., LISP (RFC 6830), VXLAN (RFC 7348), etc.). In many overlay deployments, the underlay network belongs to a different administrative domain and thus its details (e.g., topology, configuration) are hidden to the overlay network administrator (see fig. 1.3).

Typically, overlay edge nodes are connected to the underlay network via several links. Even though edge nodes have no control over the underlay routing, they can distribute the traffic among the different links they use to connect to it. Edge nodes can use overlay control plane protocols (e.g., LISP) to coordinate traffic balancing

policies across links. However, a common problem is how to find best/optimum per-link policies at the edge such that the global performance is optimized. An efficient use of edge nodes links is critical since it is the only way the overlay operator can control –to a certain extent– the traffic path over the underlay network.

Overlay operators can rely on building a model of the underlay network to optimize the performance. However, building such a model poses two main challenges. First, neither the topology nor the configuration (e.g., routing policy) of the underlay network are known, and thus it is difficult to determine the path that each flow will follow. Second, mathematical or theoretical models may fall short to model such a complex scenario.

Similar scenarios has been addressed before in the past (e.g., [13]) using network optimization techniques. ML techniques offer a new tool to model hidden networks by analyzing the correlation of inputs and outputs in the system. In other words, ML techniques can model the hidden underlay network by means of observing how the output traffic behaves for a given input traffic (i.e., f(routing policy, traffic) = performance). For instance, if two edge node links share a transit node within the -hidden- underlay network, ML techniques can learn that the performance decreases when both of those links are used at the same time and therefore recommend traffic balancing policies that avoid using both links simultaneously.

1.4.2 Resource Management in an NFV scenario

This use-case shows how the KDN paradigm can also be useful in the context of Network Function Virtualization (NFV). NFV [14] is a networking paradigm where network functions (e.g., firewalls, load-balancers, etc.) no longer require specific hardware appliances but rather are implemented in the form of Virtual Network Functions (VNFs) that run on top of general purpose hardware.

The resource management in NFV scenarios is a complex problem since VNF placement may have an important impact on the overall system performance. The problem of optimal Virtual Machine (VM) placement has been widely studied for Data Center (DC) scenarios (see [15] and the references therein), where the network topology is mostly static. However, in NFV scenarios the placement of a VNF modifies the performance of the virtualized network. This increases the complexity of the optimal placement of VNFs in NFV deployments.

Contrary to the overlay case, in the VNF placement problem all the information is available, e.g., virtual network topology, CPU/memory usage, energy consumption, VNF implementation, traffic characteristics, current configuration, etc. However, in this case the challenge is not the lack of information but rather its complexity. The behavior of VNFs depend on many different factors and thus developing accurate models is challenging.

The KDN paradigm can address many of the challenges posed by the NFV resource-allocation problem. For example, the KP can characterize, via ML techniques, the behavior of a VNF as a function of the collected analytics, such as the traffic processed by the VNF or the configuration pushed by the controller. With this model, the resource requirements of a VNF can be modeled by the KP without having to modify the network. This is helpful to optimize the placement of this VNF and, therefore, to optimize the performance of the overall network.

1.4.3 Knowledge extraction from network logs

Operators typically equip their networks with a logging infrastructure where network devices report events (e.g., link going down, packet losses, etc.). Such logs are extensively used by operators to monitor the health of the network and to troubleshoot issues. Log analysis is a well-known research field and, in the context of the KDN paradigm, it can also be used in networking [16]. By means of unsupervised learning techniques, a KDN architecture can correlate log events and discover new knowledge. This knowledge can be used by the network administrators for network operation using the open-loop approach, or to take automatic decisions in a closed-loop solution. These are some specific examples of Knowledge Discovery using Network Logging and unsupervised learning:

- Node N is always congested around 8pm and Services X and Y have an above-average number of clients.
- Abnormal number of BGP UPDATES messages sent and Interface 3 is flapping.
- Fan speeds increase in node N when interface Y fails.

1.4.4 5G mobile communications networks

The fifth generation of mobile communications networks will provide higher data rates, lower latencies, among other advances together with an important update of the network [17]. The 5G network is, by design, a Wireless SDN (WSDN), which offers a flexible network architecture, required by the specifications of 5G. Moreover, 5G is complemented by the use of Network Function Virtualization (NFV), to increase the flexibility of the 5G network and to create virtual networks over the same physical network. Within this context, KDN can be easily applied as in a conventional SDN+NFV network.

Additionally, 5G networks require novel technical solutions in which the KDN paradigm can also be helpful. The high scalability required makes necessary the design of intelligent routing algorithms for a large number of users, especially when these users are mobile [17]. The design of reliable handoffs, as well as the design of dynamic routing algorithms may take advantage of the data collected to predict the user movement

to increase the performance of these algorithms. Moreover, this can also be used to increase the efficiency of the beam-steering techniques, which will facilitate the increase of the throughput in the physical layer.

1.4.5 Network Planning

Moreover, these networks models can also be used to update the network to future needs, in a process usually known as network planning. The objective of network planning is that in the long run the network meets the requirements of the network operator (and its subscribers, if any), that is to plan ahead to prevent potential bottlenecks, packet losses or performance drops [18]. The models developed in the KDN architecture can be used to estimate the network capacity and forecast future requirements. As a simple example, KDN can learn the relation between the number of clients (or the number of services) and the load and thus, accurately estimate when a network upgrade is required.

Over time, network deployments typically have to face an increment in traffic load (e.g., higher throughput) and service requirements (e.g., less latency, less jitter, etc). Network operators have to deal with such increments and prepare the network in advance, in a process usually known as network planning. Network planning includes designing the network topology, selecting the specifications of the network hardware and deciding the traffic policies that distribute the traffic over the network. The objective of network planning is that in the long run the network meets the requirements of the network operator (and its subscribers, if any), that is to plan ahead to prevent potential bottlenecks, packet losses or performance drops [18].

Network planning techniques commonly rely on computer models managed by experts that estimate the network capacity and forecast future requirements. Since this process is prone to errors, network planning typically results in over-provisioning. A KDN architecture can develop an accurate network model based on the historical data stored in the analytics platform. As a simple example, KDN can learn the relation between the number of clients (or the number of services) and the load and thus, accurately estimate when a network upgrade is required.

1.5 Challenges

The KDN paradigm brings significant advantages to networking, but at the same time it also introduces important challenges that need to be addressed. In what follows we discuss the most relevant ones.

New ML mechanisms: Although ML techniques provide flexible tools to computer learning, its evolution is partially driven by existing ML applications (e.g., Computer Vision, recommendation systems, etc.). In this context the KDN paradigm

represents a new application for ML and as such, requires either adapting existing ML mechanisms or developing new ones. Graphs are a notable example, they are used in networking to represent topologies, which determine the performance and features of a network. In this context, only preliminary attempts have been proposed in the literature to create sound ML algorithms able to model the topology of systems that can be represented through a graph [19]. Although such proposals are not tailored to network topologies, their core ideas are encouraging for the computer networks research area. In this sense, the combination of modern ML techniques, such as Q-learning techniques, convolutional neural networks and other deep learning techniques, may be essential to make a step further in this area.

Non-deterministic networks: Typically networks operate with deterministic protocols. In addition, common analytical models used in networking have an estimation accuracy and are based on assumptions that are well understood. In contrast, models produced by ML techniques do not provide such guarantees and are difficult to understand by humans. This also means that manual verification is usually impractical when using ML-derived models. Nevertheless, ML models work well when the training set is *representative* enough. Then, what is a *representative* training set in networking? This is an important research question that needs to be addressed. Basically, we need a deep understanding of the relationship between the accuracy of the ML models, the characteristics of the network, and the size of the training set. This might be challenging in this context as the KP may not observe all possible network conditions and configurations during its normal operation. As a result, in some use-cases a training phase that tests the network under various representative configurations can be required. In this scenario, it is necessary to analyze the characteristics of such loads and configurations in order to address questions such as: does the normal traffic variability occurring in networks produce a representative training set? Does ML require testing the network under a set of configurations that may render it unusable?

New skill set and mindset: The move from traditional networks to the SDN paradigm has created an important shift on the required expertise of networking engineers and researchers. The KDN paradigm further exacerbates this issue, as it requires a new set of skills in ML techniques or Artificial Intelligence tools.

Standardized Datasets: In many cases, progress in ML techniques heavily depends on the availability of standardized datasets. Such datasets are used to research, develop and benchmark new AI algorithms. And some researchers argue that the cultivation of high-quality training datasets is even more important that new algorithms, since focusing on the dataset rather than on the algorithm may be a more straightforward approach. The publication of datasets is already a common practice in several popular ML application, such as image recognition[2]. In this thesis, we advocate that we need similar initiatives for the computer network AI field. For this reason, all

[2]Imagenet database: http://www.image-net.org/

datasets used in this thesis are public and can be found at [12]. This datasets have proven useful in routing and VNF experiments, it is our hope that help kick-off a community contributing with larger datasets.

Summary: We advocate that in order to address such important challenges and achieve the vision shared in this thesis, we require a truly inter-disciplinary effort between the research fields of Artificial Intelligence, Network Science and Computer Networks.

1.6 Thesis scope

The KDN paradigm opens a wide range of new applications in the networking field in which ML tools can play an important role. In this section, we define the scope and the specific objectives of this work.

The main objective of this thesis is to settle the basis for the use of ML techniques in the networking field and to demonstrate its feasibility. To do so, we focus on performance optimization problem based on a network model. In other words, we use ML-tools to train a "network performance simulator". We have divided this objective in three main areas: network performance modeling with simple traffic, traffic characterization and ML-based traffic forecasting, and advanced network elements characterization.

1.6.1 Network performance modeling

The main objective of this thesis is to validate the hypothesis that the performance of a network can be modeled by using ML techniques. Specifically, we focus on modeling the average end-to-end delay of a network, knowing only the traffic matrix (ingress, egress), and assuming a constant topology and configuration. Indeed, under these assumptions, the delay experimented by the different source-destination flows depends only on the amount of traffic that goes through the network.

In chapter 3, we explore the possibility of using a priori knowledge of the network behavior to design a ML-model able to capture characteristics of the network. One of the drawbacks of traditional ML models is that the resulting models are not understandable, in other words, the trained model cannot be analyzed to obtain more information of the modeled system. The first objective is to evaluate if it is possible to build a model able to obtain more information of the system.

The next step is to evaluate if networks can be modeled using ML techniques. As the starting point, we focus only on the network modeling, so we simplify as far as possible the representation of the traffic.

The objectives of this experiment are two: 1) ML-based model are clearly faster to execute than traditional simulation-based models, so they can be used to optimize networks in a faster way. The main disadvantage is that we need lots of data and time in the training period. 2) From a research point of view, the fact of validating the general hypothesis on the use of ML techniques to characterize computer networks opens a wide range of more specific network applications.

1.6.2 Traffic characterization and forecasting

In the firsts chapters of the thesis, we focus only on modeling the network for simple traffic. Afterwards, we focus on the traffic. The behavior of the network, do not only depends on its topology and configuration, but also on the traffic characteristics. Moreover, if we are able to forecast the traffic, we will be able to predict the behavior of the network, which will simplify its management. The objective is to validate that, with more complex traffic, we need more detailed information of it to be able to model the network performance. We also want to focus on other performance metrics, such as the jitter or the number of packets lost.

1.6.3 Complex network elements characterization

The last objective of the thesis, is to evaluate the estimating capabilities when modeling complex elements. The NFV paradigm makes networks more flexible by using Virtual Network Functions (VNF) instead of dedicated hardware. The main advantage is the flexibility offered by these virtual elements. However, the use of virtual nodes increases the difficulty of modeling such networks. This problem may be addressed by the use of ML techniques, to model or to control such networks. As a first step, in order to validate this approach, we focus on modeling the performance of single VNFs as a function of the input traffic.

1.6.4 Thesis outline

The remainder of this thesis is organized as follows. Before detailing the contributions of this work, we review in Chapter 2 the state of the art on different network topics related to this thesis, we briefly introduce the machine learning research area, and we review similar approaches that use ML techniques in the networking field. In Chapter 3 we explore two different approaches to model the end-to-end delay of a network: we compare a black-box model based on Artificial Neural Network (ANN) with a model inspired on traditional queuing-based models. Thereafter, we explore how these models are able to learn the delay of the network under different traffic environments in Chapter 4. Specifically, we explore different traffic intensities, different topologies, different network sizes and different routing configurations .

Chapter 5 explores how ML techniques can be used to improve the accuracy of traditional forecasting tools. In particular, we focus on improving the detection and prediction of outliers when forecasting traffic . The modeling of complex elements, such as VNFs is explored in Chapter 6, in order to evaluate the possibility of modeling complex and non linear elements. Chapter 7 concludes the dissertation with a summary of the lessons learned and paths that could be explored in future research endeavors.

Chapter 2

State of the art

In this chapter, we review the basic research topics related to this thesis, as well as we introduce different approaches similar to KDN. We start by introducing the field of Machine Learning (ML), which is the base of the Knowledge plane in the Knowledge-Defined Networking (KDN) architecture. On the next sections, we focus on the state of the art related to the networking area. We start by presenting classic and state-of-the-art network models, in order to compare the advantages and disadvantages of using ML to model networks. Right after, we review the state of the art of traffic models and traffic prediction techniques, which are needed to represent and to forecast the traffic in our models. Finally, we review the most relevant contributions which use ML in the networking field.

2.1 Machine Learning

Machine Learning (ML) is a field of computer science, the objective of which is to study and develop algorithms that are able to "learn". The "learning" concept refers to the ability of this algorithms to generalize different behaviors by only using information from training examples. In traditional software, the information needed to generalize this behavior is hard-coded in the program, whereas the code of ML algorithms defines the ability to learn, which can be used to generalize many behaviors. In other words, the main difference between traditional software and machine learning approaches is that, in machine learning algorithms, the output of the execution depends on the training phase of the software, therefore, the same algorithm can produce different outputs depending on the training data used.

Machine Learning techniques are used in a wide range of applications: image processing, voice recognition, search engines, intelligent personal assistants, self-driving cars, video games, However, there are few applications on the networking field, which we review in section 2.4.

As stated above, data is the key point on machine learning. Data is structured usually in features, which are real value representing some characteristic on the model.

Choosing relevant features and representing them in an appropriate way is an important challenge in all ML applications [20]. For example, it is difficult for ML to understand discrete labels as features, there are other mechanism to represent them, such as one-hot representations.

The training is the process in which the model learns the best parameters to minimize the error. To improve the generalization capabilities of the model, usually the dataset is divided in three sets: the training set, the validation set and the test set. The training set and the validation set are used in the training phase: the training set is used by the learning algorithm to learn the best model parameters, whereas the validation set is used to explore different ML configuration parameters and to chose the optimal. Finally, the test dataset is used to give an independent metric of the performance of the model.

Note that the focus of this thesis is not the development of new ML techniques or algorithms. In this section, we only introduce the concepts of ML used in this work. More information about ML techniques can be found in [21].

2.1.1 Machine Learning Approaches

There are different ML approach, each one useful for different applications. ML techniques can be divided as a function of the output they produce: Classification and Regression models, and as a function on the data they use to learn: Supervised or Unsupervised techniques.

On the one hand **classification models** are used to classify inputs example into different categories. In these models, the output features usually are computed as values between zero and one, representing the probability of this example of being in a particular group. The higher value is chosen as the output category.

On the other hand, **Regression models** are used to learn a continuous functions, which depends on the input features. A single ML model can be used to model one or multiple continuous functions, which increases the learning complexity, but usually it is less expensive to train a big model than N smaller models.

If we consider the other division, with **Supervised** techniques, the training data consist on pairs of input and output features, and the training algorithm infers the relation among them. With **Unsupervised** techniques, data is unlabeled, and the ML has to find relationships on the data to describe the model.

2.1.2 Machine Learning techniques

In this subsection we briefly describe the more relevant ML techniques, focusing on ANNs, which are used in most of the experiments performed in this thesis.

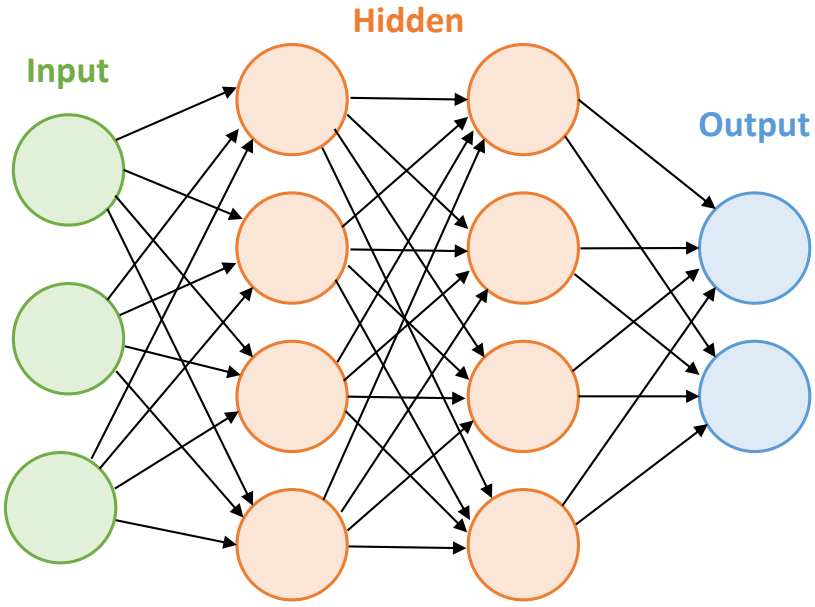

FIGURE 2.1: KDN planes

Regression techniques are statistical models, which uses traditional approaches to build continuous models. Linear models are linear combinations of all inputs features. If different powers of the input features are used, we refer to it as polynomial regression. The main advantage of linear and polynomial models is the low amount of time needed to train it.

Support vector machines (SVMs) are a supervised learning technique which divides the data in different categories, by creating different hyperplanes, Kernel functions are used to divide the data into these different planes

Artificial Neural Networks

Artificial Neural Networks (ANNs) are a ML model inspired in the behavior of biological neurons and their interconnections. Each unit or neuron performs a simple action based on multiple input signals to produce a single output. The interconnection of many neurons is known as the network. Biologic neural networks do not follow any particular structure, but ANNs usually follow a layer-based structure in which the first layer is known as the input layer, the last layer is known as the output layer and the layers in between are known as hidden layer. When there are more than one hidden layer, they are also referred as Deep Learning. Fig. 2.1 shows an example of one ANN with 3 neurons in the input layer, two hidden layer with 4 neurons in each layer and two neurons in the output layer.

Artificial Neural Networks (ANNs) were first proposed in the middle of 20th century as a biologically inspired learning algorithm. In 1975, the backpropagation algorithm was first introduced, which made possible to train multiple layers networks [20]. However, the main revolution in ANN field was in the 21th century, in which the advances in computing and GPU made possible to train complex network in a reasonable time.

Artificial Neural Networks (ANNs) are able to model complex non-linear systems, and for this reason is one of the most used ML technique. However, the main trouble when using Artificial Neural Networks (ANNs) is the huge quantity of hyper-parameters to choose, in order to be able to learn in an efficient way. The paramaters of the ANN are the values of the models that are used to compute the output, the hyper-parameters are the parameters that changes the learning process and the fitting capabilities, but that are not part of the resulting model.

Mathematically, the operations performed in an ANN can be recursively defined as:

$$y_i = f_i \left(\sum_{\forall j} W_{i,j} \cdot x_j + b_i \right)$$

where f_i is the activation function, $W_{i,j}$ of this layer and b_i are the weight of this layer. y_i stands for the output of the neuron, which is the input of the next layers except for the output layer, and x_i is the input of the neuron, which comes from the previous layer or the input. The "learning" phase consist on finding the optimal $W_{i,j}$ and b_i parameters for each layer. The optimal function is defined as the function the output of which minimize a certain cost function, which for regression problem is usually the **mse!** (**mse!**). This optimization is perform using SGE (Stochastic Gradient Descent) techniques, or different optimization of these techniques. All of these techniques are based on performing small steps to minimize the cost function while evaluating the function to decide where to go.

In the following list we briefly describe the set of the most relevant hyper-parameters:

- **Activation function:**
  It is the function performed on the weighted sum of the input values of a neuron. The most common functions are: linear, sigmoid, rectified (ReLu) and hyperbolic tangent.
- **Learning Rate:**
  It defines the step done in each iteration to minimize the cost. A low learning rate increases the training time, whereas a high learning rate may jump over the minimum.
- **L2 regularization:**
  To avoid over-fitting, we can define a parameter that increases the cost of high

neuron parameters. The optimal parameter completely depends on the application and has to be found empirically.

- **ANN size:**
  It is not a hyper-parameter, but the size and the topology of the network (number of layers and neurons per layer) has to be chosen before the training, and they determine the training time and fitting capabilities.
- **Other:**
  Other parameters have more impact upon the training time than upon the error. For example: the optimization algorithm, the size of the epoch (which is defined as the number of samples processed per each iteration), the maximum number of iterations, ...

2.2 Networking: Novel networking paradigms and network modeling approaches

In this section, we introduce two relevant topics. First, we introduce two novel network paradigms, Software-Defined Networking (SDN) and Network Function Virtualization (NFV), which are changing the networking field and enables the KDN proposal. Second, we review the different network models approaches done throughout history.

2.2.1 Software-Defined Networking and Network Analytics

Software-Defined Networking (SDN) was created to overcome the limitations of traditional networks [22]. SDN separates data plane and the control plane to help in the control and management of the network; and it offers a centralized point of control and the the ability to program the network elements. Both centralization and programmability are novelties offered by the SDN paradigm that are explored to improve the performance of networks.

The architecture of SDN consist on 5 components: SDN Datapath, SDN Controller, SDN Application, SDN Control to Data-Plane Interface (CDPI) and SDN Northbound Interfaces (NBI). The SDN Datapath is the logical view of the network that the traffic sees, and it may correspond or not to the physical network. The SDN Controller is the logically centralized unit in charge of communicating with the network elements in their own language, translating the instructions received by SDN Application or human administrators. This controller is connected to every network element in the network. SDN Application are programs that control and take decisions about the behavior of the network. These applications can be as complex as

needed, and can their function is usually to simplify and to automatize the management of the network. However, since they are the upper-level component in the SDN architecture, they are the less explorer component in the SDN architecture. Finally, the CDPI and the NBI are the interfaces between the controller and the application and datapath respectively. Each interface uses a predefined language to communicate; for example, Open Flow (OF) is standardized and the most used control to data-plane language.

The concept of Network Analytics (NA) is used to define the new set of network monitoring techniques, which make possible to collect real-time packet and flow-granularity information, as well as configuration and network state monitoring data, to a centralized NA platform [3]. This is mainly enabled by the "softwarization" of the network, current network data plane elements, such as routers and switches, are equipped with improved computing and storage capabilities. This information is fundamental to understand the network functioning in order to improve the control and the management of the network. ML tools can use this relevant data to facilitate this work.

2.2.2 Network Function Virtualization

A different research area, but closely related to SDN is the Network Function Virtualization (NFV) [23]. It proposes the use of virtualalization to create Virtual Network Function (VNF), which can be compared to VM in the networking field. Example of VNF are load balancers, firewalls, intrusion detection devices and WAN accelerators (Wide Area Network). The main advantage of using VNF is the flexibility offered by virtualization, as occurs in Data Centers and VMs. Moreover, this VNF can be easily deployed or moved to a different point of the network; and the software of the VNF can be easily updated. A VNF can be implemented in a traditional non-SDN network, but it is when SDN and NFV are working together when a great benefit can be obtained. The centralized point of observation in SDN can be used to optimize the placement of the VNF, and the programmability inherent in SDN is a perfect complement to configure the different VNFs.

2.2.3 Network Modeling

Computer networks are complex system, which are difficult to operate. Modeling these systems is an important challenge that easies the operation of computer networks. Different approaches have been followed to achieve this objective, from analytical models to advanced simulation tools [24]. In this chapter we summarize the most relevant approaches.

2.2.4 Markov chains and queuing theory

The Markov chain theory [25] is widely used to model any kind of processes described by discrete events, such as the generation and the sending of communication packets. Mathematically, a Markov chain is defined as a memoryless stochastic process, in which the probability of being in a certain state of the chain does not depend on the time. This property is the basis of the queuing theory, which models the behavior of the packets sent through a queue.

A fundamental problem in the queuing theory is the modeling of the arrival and departure rates. Different arrival and departure distributions produce different models, which implies that we do not only need to model the queue, but also we need to model the process that goes through the queue. With real traffic, this arrival and departure rate is translated as the inter-arrival distribution, the packet size distribution and the capacity of the different links of the network.

Commonly studied arrival and departure distributions are Markov distributions and Deterministic distributions. Note that even if they are not memoryless distributions, they can be used to create analytical models for single queues. These distributions gives well-known models, such as M/M/1, M/D/1, and other variations with finites queues or with more than one output.

These models have been extended to model networks of nodes, i.e. queuing networks [26]. Example of these theories are: Jackson Networks, Gordon-Newell theorem, Mean value analysis, Buzen's algorithm, Kelly network, G-network, BCMP network [27, 28]. As with single queues model, these theoretical models need to assume certain probabilistic of the traffic, which it is translated as assuming certain relationships among nodes and how they route packets.

The main disadvantage of these analytical models is the need to assume certain traffic properties, as well as how the traffic is routed inside the network.

2.2.5 Network Calculus

A different approach is Network Calculus, which is defined as "*a set of mathematical results which give insights into man-made systems such as concurrent programs, digital circuits and communication networks*" [29].

Network Calculus combines different network constrains, and traffic arrival and departure functions by using convolution under min-plus algebra. Example of these constrains are the capacity of the links, the congestion control protocol used and the carried traffic. The min-plus algebra, sometimes referred as tropical geometry, is an area of mathematics which, by modifying some common operations, is able to transform non-linear problems into linear and tractable problems.

Computer networks present strong non-linearities that can be addressed with Network Calculus. However, this theory is still stuck in the need of traffic models. In Network Calculus, these traffic models are named traffic envelopes and services curves, which models the arrivals rates and services respectively.

2.2.6 Discrete-Event Simulations

Computational models, and more concretely, discrete-event simulations, models the operation of the network in a discrete sequence of events. These simulations take advantage of of the discrete changes of state in the network, and consequently, it is only necessary to model the changes of the state in the network, i.e., the activity of the packets. The most relevant simulators are *ns* [1] and *mininet* [2] Another well-known software is Omnet++ [30], which is discrete-event framework to develop a simulator on the top of it.

These simulators typically operate at the packet level, and simplify the network protocols they simulate. This last simplification is done to reduce the simulation time, since each change of state increases the simulation time. Moreover, with this simulators, it is possible to use any source of traffic with arbitrary distributions, which is an important advantage in comparison with analytical models. However, the main drawback is the high computational cost to run a single simulation.

2.2.7 Our approach

In this thesis we advocate that, enabled by the global view and control offered by SDN and analytics techniques, ML can represent a third pillar in network modeling providing important advantages: ML is data-driven, does not require simplifying assumptions typically found in traditional network modeling, works well with complex systems (e.g., non-linearities and multi-dimensional dependencies) and, if trained well, can be general.

2.3 Traffic modeling and prediction

As discussed in the previous section, it is important to model the traffic that goes inside the network to properly model the behavior of the network. Moreover, to be able to predict the performance of the network in the future, it is necessary to forecast the traffic at that specific time. In this section, we review the state of the art on this area.

[1] https://www.nsnam.org/
[2] http://mininet.org/

2.3.1 Statistical models

Traffic models are statistical models which are able to generate random traffic based on certain assumptions. The two main assumptions of the traffic are: 1) the inter-arrival time distribution and 2) the packet size distribution. Traditionally, well-known mathematical distributions have been used, which makes possible to develop analytical models due to the properties of these distributions. Examples of these distributions are: Poisson distribution, Uniform distribution, Binomial distribution, and also, the Deterministic distribution. The most used distribution in the analytical models is the Poisson distribution, due to the memoryless property. These models may be useful for some applications, however do not accurately reproduce the actual traffic of the Internet.

The **Self-similar** process has been introduced to better model the Internet traffic [31, 32]. The self-similar model is characterized by the fact that the traffic behaves the same when viewed at different scales on time. In other words, Internet traffic exhibits same statistical properties in different time scale, which it is translated into time dependencies over long range of time scales.

One of the characteristics of the self-similar traffic is the burstiness nature of it. The presence and the intensity of these bursts is one of the parameters of this model (usually known as the Hurst parameter). The more bursty the distribution is, the more different the traffic is with respect to traditional traffic models. Heavy-tailed, or long-tailed distributions, such as the Pareto distribution are examples of self-similar processes.

2.3.2 Trace-based simulations

Trace-based simulations, actually, are not traffic models. Instead, trace-based simulations are a common technique used to experiment with more realistic traffic sources. It consists on using previously captured information of real traffic, which can includes from the time and size distribution up to addresses and protocols, which implies that we can use this information to test the network at any layer.

Traffic can be obtained from public databases, such as: [33–35], or obtained in local network, such as the traces obtained in the UPC campus [36]. This traces contains certain information from each packet, i.e., every packet is recorded, with some information. Information can be: the time stamp, the different headers of every layer, and also the content of the packet. However, due to privacy issues, the addresses and the content are usually anonymized or deleted. In some high speed links with a huge amount of traffic, this can be sampled, to reduce the size of the trace.

With the information captured in the trace, it is possible to generate synthetic traffic with the same characteristics than the captured traffic.

2.3.3 Traffic forecasting

Traffic forecasting is a widely studied field typically using time series techniques. Autoregressive Moving Average model (ARMA) and Autoregressive Integrated Moving Average (ARIMA) are linear modeling techniques based on time series analysis able to fit complex time series data. Both of them are based on an Autoregressive (AR) part, which involves representing the variable as a combination of previous values, and Moving-Average (MA), which involves modeling the error as a combination of the error of past samples. In ARIMA, if the data is not stationary, it is differentiated until it becomes stationary.

Other areas, such as logistics, present similar problems, and have proposed new forecasting techniques that can be used in the traffic forecasting field. In order to improve traditional ARMA/ARIMA techniques it has been proposed to use them in combination with other models such as Artificial Neural Networks (ANN) [37, 38]. The main reason behind this is that neural network are able to account for non-linear dependencies.

2.4 Machine Learning in the networking field

In the previous sections of the state of the art chapter, we briefly introduced the ML field as well as networking topics related to the work of this thesis. However, the main contribution of this thesis is the use of Machine Learning (ML) techniques in the networking field. In this section, we review the most relevant contributions in the networking area that make use of ML tools.

The idea of using ML techniques to the networking area is not new. However, most of the contributions are focused in the traffic analysis field, such as traffic classification and anomaly and intrusion detecting systems [39, 40]. They take advantage of the ability of ML techniques to discover a hidden structure or representation of the traffic, and then, they are able either to classify or cluster the traffic into different groups or to detect unusual traffic patterns. However, these proposals only inspects the traffic and do not consider the rest of the network.

Another interesting application of ML in the computer field is the optimal placement of Virtual Functions in Data Centers (see [41]). They usually apply deterministic algorithm to optimize to metrics: to improve the performance and to reduce the energy consumption. The fact of being able to predict the performance of the network before actually applying the changes will allow to develop novel placement techniques, and not only for VM but also for VNF.

Recently, some initiatives share the same vision stated in the introduction. A relevant one is *COBANETS*, from COgnition-BAsed NETworkS [7–9]. The main difference between KDN and COBANETS is the wide range of applications and use-cases

proposed in the KDN paradigm. From the industry perspective, there are also some projects which make use of ML techiques to optimize some points of the network. A relevant example is the Self-Driving Network introduced by JUNIPER[3].

[3]https://www.juniper.net/us/en/dm/the-self-driving-network/

Chapter 3

A theoretical inspired approach

3.1 Introduction

The application of ML to networking brings several use-cases as well as challenges. In this chapter, we make the first approach on the use of ML techniques in the networking field. Specifically, we focus on a rather fundamental problem in networking: *estimating the delays of a network*, and we compare to more traditional approach to estimate this delay. We consider a simple scenario with a set of traffic producers/consumers (overlay nodes) and forwarding nodes (underlay nodes), the overlay nodes are connected through various links to a subset of underlay nodes and can traffic-engineer packets across the different links (overlay routing policy). The underlay nodes use in all cases shortest-path routing.

The objective of this chapter is to understand if we can automatically train a model to estimate the delays of the network, given the traffic load and the overlay routing policy. Please note that in our scenario we consider the topology as a hidden variable. We evaluate a generic adaptive machine learning model, based upon artificial neural networks, and we compare it with a specific model designed to capture information of the network.

An important drawback when training a model using neural networks is that these models, in contrast to traditional network models (e.g., Markov chains), are difficult to understand and do not provide guarantees based on well-understood assumptions. This also means that manual verification is usually impractical when using neural-network models. For this reason, we also want to asses models inspired by the existing knowledge of network modeling. Specifically, we present a M/M/1-inspired ML regressor that, if accurate, would provide insights about the internal structure of the modeled network. With this, we compare the performance of both estimators.

3.2 Problem statement

In this section we describe the problem statement that this chapter aims to address. The objective is to estimate the average end-to-end delay among all pair of nodes in a computer network, as a function only of the amount traffic in the network.

In this chapter, we consider a fixed topology and configuration of the network along the experiments. This network consists on an overlay-underlay network, since it completely hides the topology of the hidden network. In the learning process we only consider the traffic matrix among overlay nodes to learn the average delay matrix among all nodes of the overlay network.

As stated in the introduction of the chapter, we first compare a black-box ML-approach, i.e., an ANN model, with a network-knowledge-aware model, defined in the next section. Both models are designed to learn and estimate the delay of the network only as a function of the traffic and the routing on the overlay network. The main difference between models is the priori knowledge of the network behavior assumed in the second model.

We also want to compare a model trained with the local information of a node of the network with a model trained with the global information of all nodes of the network. The objective is to validate that the fact of using all the information of the network makes possible to create better models of the network.

The main questions that we aim to address in this chapter are:

- Can we design a machine learning model able to use a priori knowledge of networks to easily capture the behavior and the hidden characteristics of the network?
- Can we train a neural network to estimates of the mean end-to-end delay among the nodes of the network, modeled as a black box?
- Does the learning ability of the different models depends on the network and traffic characteristics?
- Does the learning ability improves when using the global information of the network instead of the local information?

3.3 Methodology

To assess the validity of this approach, we carried out different experiments. To obtain the dataset, we have simulated (see Appendix A) an overlay-underlay network, The overlay nodes contains all the stations that generate and receive traffic and the underlay nodes holds a routing structure operating on a shortest path policy. The overlay network has 12 nodes, the underlay has 19 elements and a total of

72 links, all of them with the same capacity. The simulation has the following traffic characteristics: overlay nodes generates Poisson traffic and randomly split the traffic independently of the destination node and the underlay network carries its traffic to the destination. From the KP perspective, the ML model only sees the overlay nodes that send and receive traffic, while the underlay network is hidden.

The training, validation and test sets we use have been obtained throughout a simulation process in Omnet++ [1], implementing a network with 12 overlay nodes and 19 underlay nodes. The overlay nodes contains all the stations that generate and receive traffic and the underlay nodes holds a routing structure operating on a shortest path policy. Note that the structure of the underlay network is unknown for the learning process. The dataset consists on a set of 10,000 repetitions of the simulation under different traffic conditions. Each samples consists on 1) the traffic matrix with the amount of traffic (per second) sent among source-destination pairs of the overlay nodes, 2) the routing of the overlay, defined as the ratio of traffic that is sent through each edge link and 3) the delay matrix with the average delay among the same nodes.

Therefore, ML models are trained using as input features the traffic matrix and the routing information and the delay matrix as output features. The dataset has been slitted in three sets, the training set, the validation set and the test set, in order to choose the best parameters during the training and to estimate the model error. volume, defined as the aggregated bytes for the simulated time

3.4 Machine learning techniques

In this section, we present a new machine learning regressor based on the theoretical model for delays in queuing networks, as well as the more common ANN models used to compare their performance.

3.4.1 A M/M/1-Inspired regressor

Our machine learning regressor is based on the theoretical model for delays in queuing networks, known as Jackson networks [27, 28]. Since queuing networks commonly assume M/M/1 queuing systems, we will refer to the new predictor in the upcoming lines as the fitted M/M/1 model.

[1] www.omnetpp.org

Delays in M/M/1 queuing networks

In a M/M/1 link, the expected delay any packet would experience is described by [26] as: $D = 1/(C - \lambda)$, in which, both C, the capacity, and λ, the inter-arrival rate, are expressed as packets/s.

In a Jackson network, we assume that the expected delay is the addition of the delay of all links it goes through:

$$D = \sum_{i=0}^{M} \frac{P_i}{C_i - \lambda_i} \tag{3.1}$$

where M represents the number of links (or queues) the packet would go through until it reaches its destination, C_i the capacity at each link i and λ_i the total amount of traffic flowing through the link. Also, P_i stands for the average number of times a packet goes through link i. It is important to remind that this model is valid as long as $C_i - \lambda_i > 0 \; \forall i$.

Model definition

The fitted M/M/1 estimator is derived using a reformulation of Eq. (3.1):

$$\widehat{D} = \sum_{i=0}^{M} \frac{P_i}{C_i - \lambda_i} = \sum_{i=0}^{M} \frac{P_i}{C_i - \underline{k_i} \cdot \underline{\gamma}} = \sum_{i=0}^{M} \frac{1}{a_i - \underline{b_i} \cdot \underline{\gamma}} \tag{3.2}$$

where λi has been expressed as $\underline{k_i} \cdot \underline{\gamma}$, a dot product between $\underline{\gamma}$, all the input traffics in the network, and $\underline{k_i}$, their proportion flowing through link i. To achieve a more simplified expression, we define $a_i = C_i/P_i$ and $\underline{b_i} = \underline{k_i}/P_i$.

The optimization of this predictor consists of a traditional statistical learning process: finding the parameters a_i, $\underline{b_i}$ that minimize a cost function (error rate) over a training set. A training set consists of a set of pairs $(\underline{\gamma_t}, D_t)$, where $\underline{\gamma}$ corresponds to the amount of traffic that is being generated in the network and D the delay we observe in the targeted packets under these conditions. As cost function we will be using the MSE (Mean Squared Error) over a T-sized training set:

$$E = \frac{1}{2} \sum_{t=0}^{T} (D_t - \sum_{i} \frac{1}{a_i - \underline{b_i} \cdot \underline{\gamma_t}})^2 \tag{3.3}$$

Finding the optimal parameters of the model analytically is impracticable. Consequently, we suggest using stochastic gradient descent, a first-order optimization algorithm that operates taking successive steps proportional to the negative of the gradient of the cost function at the current point, until the optimum is found:

$$a_i^{(n)} = a_i^{(n-1)} - \alpha \frac{\partial E}{\partial a_i} \tag{3.4}$$

$$b_{i,j}^{(n)} = b_{i,j}^{(n-1)} - \alpha \frac{\partial E}{\partial b_{i,j}} \tag{3.5}$$

where α represents the step size or learning rate, the parameter that will determine the convergence to the global optimum. The partial derivatives are as follows:

$$\frac{\partial E}{\partial a_i} = \sum_{t=0}^{T} \frac{D_t - \sum_t (a_i - \underline{b_i} \cdot \underline{\gamma_t})^{-1}}{(a_j - \underline{b_j} \cdot \underline{\gamma_t})^2} \tag{3.6}$$

$$\frac{\partial E}{\partial b_{k,l}} = \sum_{t=0}^{T} \frac{D_t - \sum_t (a_i - \underline{b_i} \cdot \underline{\gamma_t})^{-1} \cdot \gamma_{t,l}}{(a_k - \underline{b_k} \cdot \underline{\gamma_t})^2} \tag{3.7}$$

Proving the convexity of the cost function guarantees that the gradient descent always finds the global optimum when a sufficiently small learning rate is selected [42]. However, for most machine learning models convexity is not ensured in their cost function, even if usually trained with convex optimization techniques. In these cases, several local minimum may arise. This challenge is normally addressed by carefully selecting and trying more cases in the cross validation process, taking a new compromise between training time and error. In this chapter we focus in an experimental evaluation, and we treat these considerations by iterating a sufficient number of times when validating each model, in the way we have described.

For the this model we have used our own implementation, consisting in the classical SGD algorithm with fixed step size and number of epochs. We have explored different parameters, as described in the next section.

3.4.2 ANN models

In the different experiments we used two different ML tools to implement the ANNs. For the first experiment, we used the scikit-learn python package [2] selecting a sigmoid activation function and a stochastic gradient-based optimization algorithm proposed by Kingma, Diederik, and Jimmy Ba [43] with a constant learning rate. We have explored different hyper-parameters, as described in the next section. For the second experiment, we used a different ML library. We trained an ANN using Pylearn2–Theano 0.7 with one hidden layer and a sigmoid activation function.

[2] www.scikit-learn.org/stable/

3.5 Experiment 1: Comparison and Evaluation of the M/M/1 model

3.5.1 Overview

In this experiment we compare the accuracy of the fitted M/M/1 model and a single-layer neural network.

For each training, the optimal parameters for the experimental M/M/1 model (number of links M and step size α) and for the neural network (regularization parameter and number of neurons in the hidden layer) are selected using a cross validation methodology. We repeat the training and validation process a sufficient number of times in each case to guarantee that the results provide at least a 5% beta-confidence. After the cross-validation selection, the final results displayed in the plots are obtained using an independent test-set. The different parameters selected for cross-validating the models are restricted to the following:

- Step sizes = (0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0)
- M = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
- Regularization parameters = (0.0001,0.001,0.01,0.1, 1.0, 1.0, 10)
- Number of neurons in the hidden layer = (2,3, 4, 5, 7, 10, 15, 20, 30, 40, 50, 60, 80, 100)

The results have been evaluated using a normalized relative error:

$$e = \sqrt{\frac{1}{T} \cdot \sum_{t=0}^{T} (\frac{\widehat{Y}_t - Y_t}{Y_t})^2} \tag{3.8}$$

where T states for the number of examples, $\widehat{Y}_t$ the predicted outcome and Y_t the expected one.

3.5.2 Results

The first experiment compares the learning capacity of the fitted model with a neural network, observing the required number of examples in each case to achieve a specific error rate. In this first experiment, and for practical purposes, we have restricted the simulations to only three active, transmitting, stations in the network. Fig. 3.1 shows the best test error rate every model has been able to achieve as a function of the training set size. The selected parameters after the cross validation selection are in this case M=8 and a step size of 0.01 for the fitted M/M/1, while 50 hidden neurons and a cost parameter of 1.0 have shown to be the best choices for

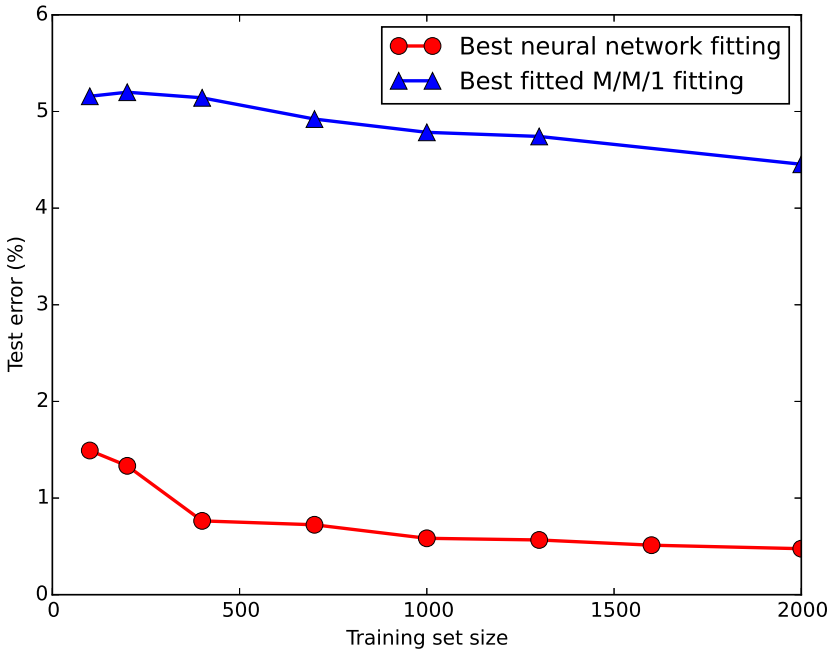

FIGURE 3.1: Evolution of the relative test error as a function of the training set size

# Active stations	Level of saturation	Best C	Best # hidden neurons	Best M	Best step size
1	1	10.0	30	3	0.1
1	2	10.0	30	8	0.0001
1	3	10.0	20	7	0.00001
1	4	10.0	7	7	0.00001
3	1	1.0	50	8	0.01
6	1	1.0	30	8	0.01
9	1	1.0	50	8	0.01

TABLE 3.1: Best cross-validated parameters

the neural network. The results clearly show how the neural network learns better and faster the inherent patterns in the data.

In the next experiments, we test how does the saturation in the links affect the predictions and how robust are the results to an increase of the number of active (transmitting) stations in our network. Fig. 3.2 and 3.3 show respectively how the best test error rate changes when we increase the number of active stations and the level of saturation in the links. For the second case, we set a reference in the third level, where at this point the links (on average) are expected to start saturating. The simulations have been developed according to the same approach described in the first experiment, and the optimal parameters for these cases are shown in Table 3.1.

Again, the M/M/1 experimental model performs poorly compared to the neural

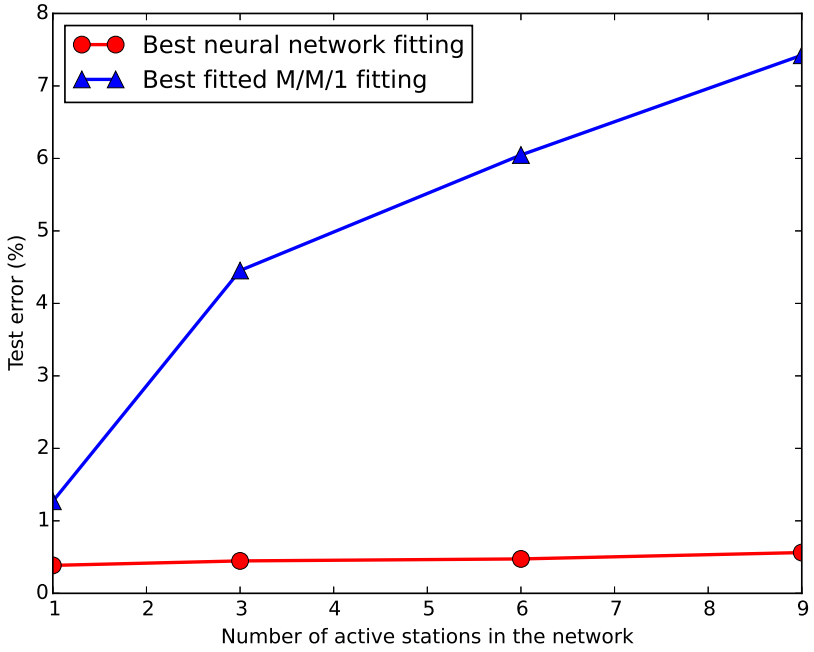

FIGURE 3.2: Evolution of the relative test error as a function of the number of actively transmitting stations in the network

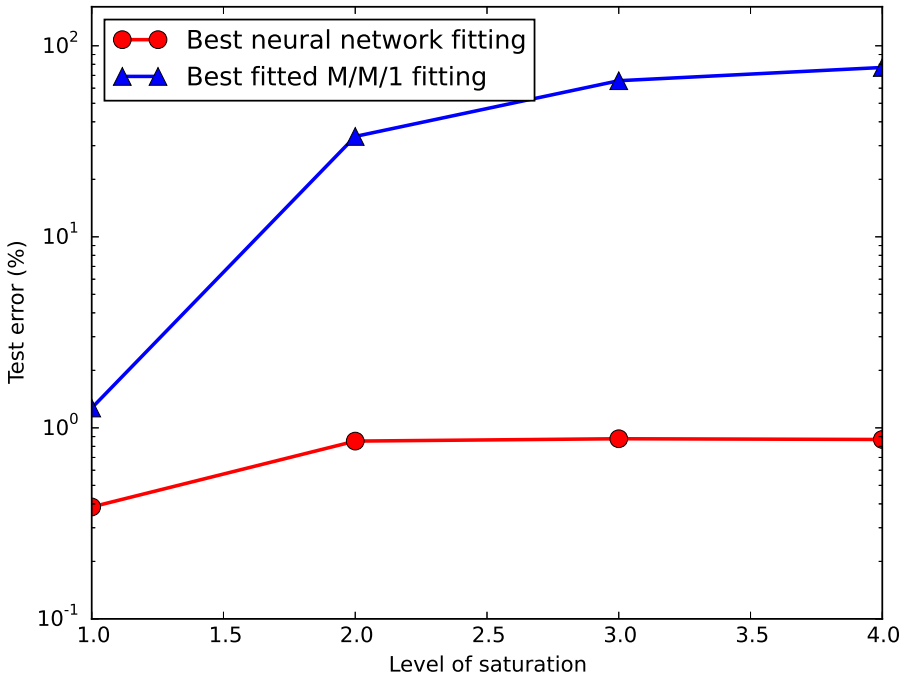

FIGURE 3.3: Evolution of the relative test error as a function of the average level of saturation in the links (logarithmic scale)

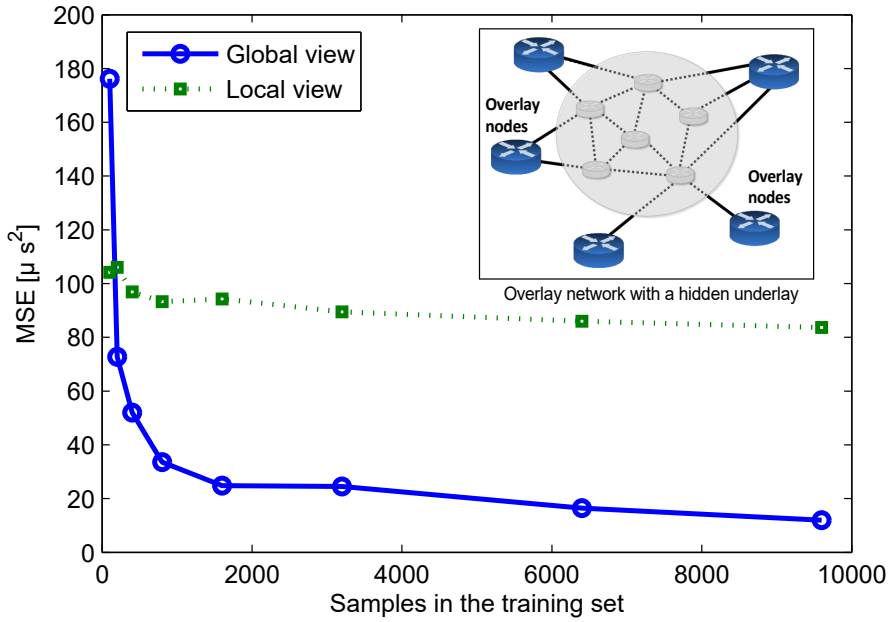

FIGURE 3.4: Prediction error (Mean Square Error) as a function of the size of the training set in an overlay-underlay scenario, both for global and local view.

network, with an additional significantly bad tolerance to load increase in the network. The poor performance of the experimental M/M/1 model can be attributed to the lack of fulfillment in our network of Jackson's assumptions, such as overall arrivals independence or exponential distribution for packet lengths in all successive routers.

3.6 Experiment 2: Global vs local view modeling

3.6.1 Overview

Once we have shown that neural network models outperform the M/M/1 regressor, we want to verify than the use of the global information of the network is needed to better modeling the end-to-end delay of the network. We also want to verify the size of the training set needed to train the model.

To do so, we train two different ANN models. The local view model only uses the information obtained from one node to predict the delays observed from that node. The global view model has the same structure than the used in the previous experiment. We train these models with different number of samples, which makes possible to characterize the effect of the size of the dataset in the training process.

3.6.2 Results

Fig. 3.4 shows the error (the accuracy) of the model as a function of the training set size (solid line). This error represents how accurately the model predicts the delay when the routing and traffic is known, but not the topology. As shown by the figure, the relative error is roughly 1% when using 6,400 training samples, equivalent to a mean square error of 20 μs^2. In addition to this, fig. 3.4 also shows (dashed line) when the model is trained only using local information. The main reason behind this experiment is that we aim to validate the main hypothesis stated in this paper: ML applied to a global view renders better results than when only local information is available. For this, each overlay node is trained only with local traffic, routing and delay and as the results show, the accuracy in this case is strongly degraded. This is because the delay between two nodes depends on the state of the queues of the underlay network, which in turns depends on the total traffic of the network.

3.7 Conclusions

In this chapter we have studied the performance of two different models to characterize the delays in a network given the traffic load. We have demonstrated that simple neural network based estimators performs remarkably well. However, these estimators models the network as a black-box and do not provide insides of the behavior of the network. For comparison, we implemented a M/M/1-inspired estimator, which could provide more information of the network, but it has leaded to poor results. Moreover, we have demonstrated that a model trained with the complete information of a network outperforms the models trained with a local view of the network. Finally, we also verified that we need thousands of samples to train the model for the network used in this chapter.

Chapter 4

Network modeling using ANN

In the KDN context, learning techniques can be used to provide automatic control of the network via the SDN controller thanks to the network monitoring information obtained via the NA platform. This new networking paradigm is known as Knowledge-Defined Networking (KDN) [44].

Under the KDN paradigm there are a wide variety of use-cases for taking advantage of ML techniques in computer networks. Among all such potential use-cases, in this chapter we focus in a single one: *modeling of network performance using neural networks*. The main reason for this is that network modeling is central to many network operations, particularly in the field of network optimization. Typically network optimization algorithms require of a network model over which the optimization techniques operates in order to find the best element (e.g., [45–47]). chapter In this chapter we aim to answer the following question: *Can neural networks accurately model the delay of a computer network as a function of the input traffic?* For this we assume the network as a black-box that has as input traffic and as output delays. Following this approach we experimentally evaluate the accuracy of the neural network when estimating the end-to-end delay of traffic, we also compare it to other well-known regressors and we study the effect of fundamental network characteristics (topology, routing, etc) in the accuracy of the delay estimates.

The question that we aim to address in this chapter is fundamental to network modeling. Indeed both analytical (e.g., queuing models) and computational models (e.g., simulators) are well-known techniques used to estimate the performance of a network based on its input traffic. In this chapter we posit that neural networks can represent a third pillar in the area of network modeling, providing relevant advantages towards traditional techniques. At the best of our knowledge this is the first attempt to model a computer network using neural networks.

Indeed, in this chapter we advocate that neural networks represent a third pillar in the field of network modeling. Neural Networks can efficiently complement existing analytical and computational techniques, particularly in network optimization scenarios. The main advantages of neural networks with respect to existing techniques are:

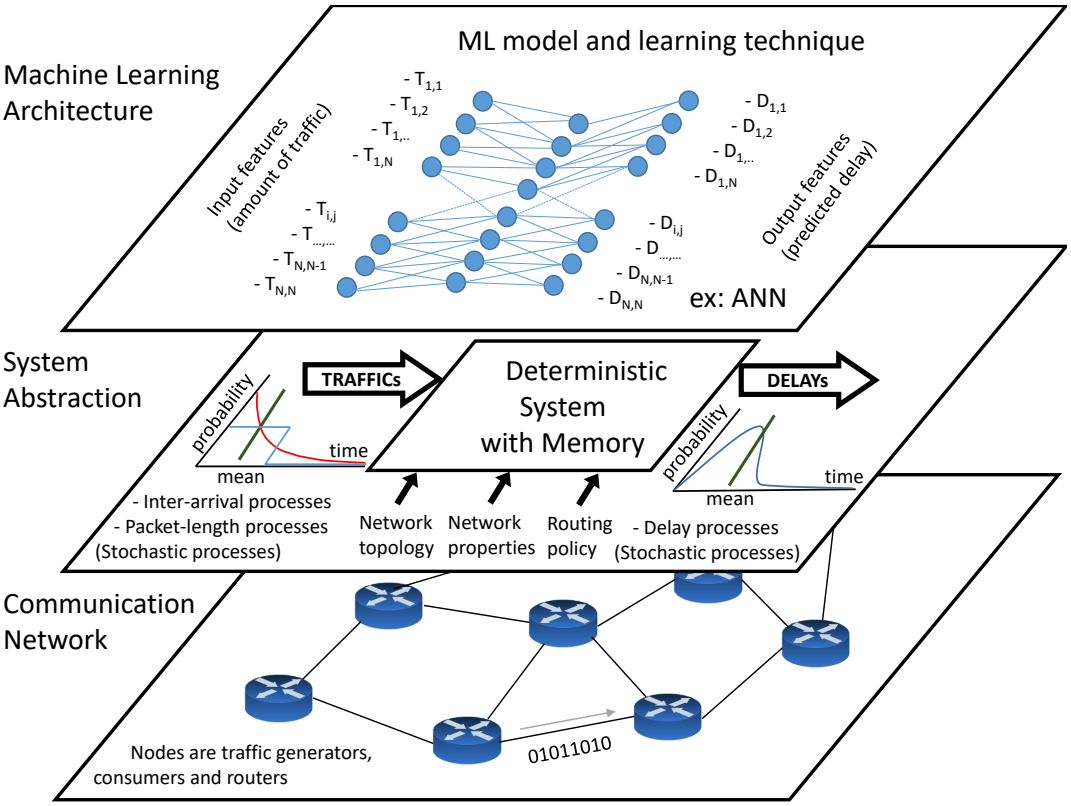

FIGURE 4.1: Graphical representation of the problem statement addressed in this chapter.

4.1 Problem Statement

In this section we describe the problem statement that we aim to address in this chapter. Figure 4.1 summarizes the problem statement using three layers. The bottom layer represents the real-world physical network infrastructure that has certain fundamental characteristics, such as topology, size, routing, etc.

The middle layer represents the system abstraction where the network is assumed as a black box, traffic ingresses the box and egresses it with a certain average delay. The traffic is described by stochastic distributions, both the inter-arrival process and the packet length process. These stochastic processes are combined in the network, which is a deterministic complex system with certain properties (topology, routing, etc) and memory when random process such as physical errors are not taken into account.

Finally, the top layer represents the neural network that models the computer network performance. The neural network is able to produce estimates of the average end-to-end delay for all pairs of nodes considering the input traffic as a traffic matrix [ingress, egress]. The network characteristics (routing, topology, etc) are hidden from the neural network, and hence the neural network is trained only for one particular configuration of the network infrastructure that is, a certain topology, routing,

etc.

The main questions that we aim to address in this chapter are:

- Can we train a neural network to produce accurate estimates of the mean end-to-end delay for all pairs of nodes considering the input traffic matrix (ingress, egress)?
- Which is the impact of fundamental network characteristics (topology, routing, size, traffic intensity) with respect to the accuracy of the neural network?
- How does the accuracy of a neural network compare to other well-known regressors?

4.2 State-of-the-art

The main goal of this chapter is to understand if a neural network can be used for network modeling, specifically to estimate the average delay of a computer network as a function of the input traffic matrix.

Network modeling is a well-established field that provides techniques which are central to a wide range of communication functions, for instance in network optimization. In order to optimzie models are used to estimate the performance of configurations, then an optimization algorithm searches the configuration space using these models to find the best possible one, that is a configuration for which the model estimates the best performance. Notable examples of this are in the area of Traffic Engineering (TE) [48], where models of the network are used to find routing configurations that achieve a certain target performance.

In the field of network modeling there are to fundamental approaches: analytical and computational models (simulation) techniques. As for analytical techniques, Markov chain theory has been widely used in queuing theory to model the behavior of a single queue by assuming certain stochastic proprieties of the job arrival and job completion processes (ex. M/M/1, M/D/1...). These models have been extended to model networks of nodes, i.e. queuing networks [49]. Example of these theories are: Jackson Networks, Gordon-Newell theorem, Mean value analysis, Buzen's algorithm, Kelly network, G-network, BCMP network. Computational models are also another popular technique to model the behavior of networks. Typically simulators operate either at packet or flow level and simplify the network protocols they simulate.

Machine Learning mechanisms have been used in the field of communications, such techniques have been used extensively in the area of traffic analysis [50], network security [51] and root-cause analysis [52]. In addition, some works propose the use of Reinforcement Learning techniques for routing optimization [53]. However and

at the best of our knowledge, this is the first attempt to model the network as a black-box using neural networks.

- **Accurate in complex scenarios:** Typically analytical techniques are based on strong simplifying assumptions of the underlying networking infrastructure: this is because they need to be tractable. On the other hand simulations can model complex behavior, but this comes at a high development and computational cost. ML techniques and particularly neural networks work very well with complexity (e.g, non-linear behaviors) and high-dimensionality scenarios. As such, well-trained neural networks can model computer networks without making any simplifying assumptions while providing accurate estimates.
- **Fast and lightweight:** Neural networks require important computational resource for training, but once trained they are fast and lightweight. Indeed they can produce estimates of the performance of the network in one single step and require very little resources to run. This represents an important advantage particularly in front of simulators that require important computational resources to run and might be slow.

The main disadvantage of neural networks is that they are data-driven techniques, and as such require large training sets as well as computational resources for the learning phase.

4.3 Methodology

In this section we detail the methodology used to understand if a neural network can estimate the average delay of a computer network as a function of the traffic matrix.

4.3.1 Overview

Figure 4.2 shows an overview of the methodology. In order to experimentally analyze the accuracy of the neural network we generate different training set by means of simulations, in each set we change different network characteristics: traffic distribution, traffic intensity, topology, size and routing policy and measure the average delay.

Once we have generated the data set we use them to train a set of regressors including a neural network, then we evaluate its accuracy using the cross-validation technique. We split the data set into three sets, the training set with 60% of the samples, the validation set with 20% of the samples and the test set with the remaining 20% of the samples. The training set is used to optimize the ML model, the validation set

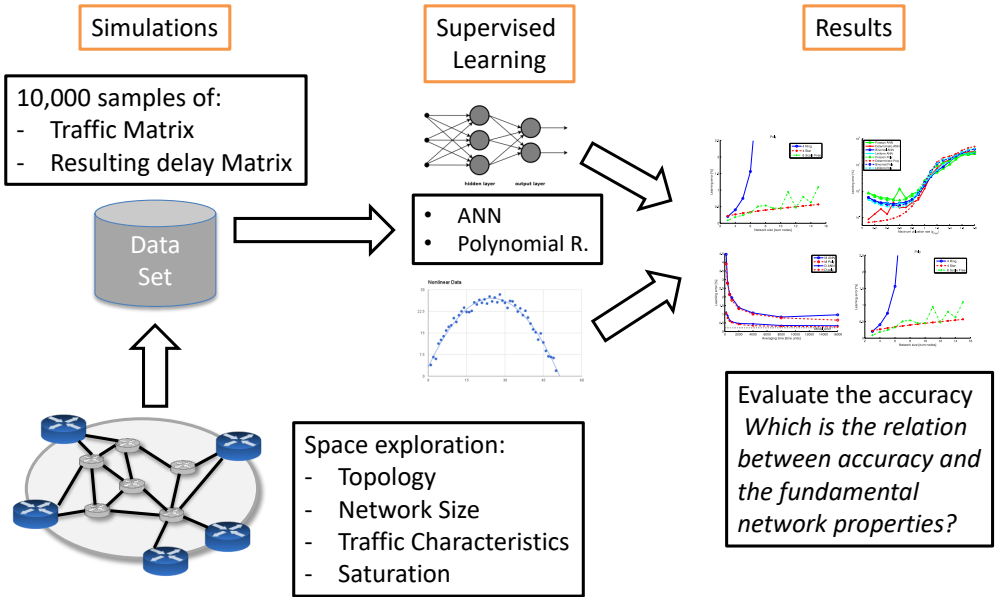

FIGURE 4.2: Scheme of the methodology followed in this work

is used to evaluate the model during the training phase and the test set is used to provide an independent evaluation of the performance. With this we compare the average delay estimated by the regressors with the one measured from the simulator. Ultimately, we want to understand both the accuracy of the neural network and its relation with fundamental characteristics of networks: traffic distribution, traffic intensity, topology, size and routing policy.

4.3.2 Network Simulations

In order to generate the data-set we use the Omnet++ simulator (version 4.6), in each simulation we measure the average end-to-end delay during 16k units of time for all pairs of nodes. The transmission speed of all links in the network is set to 10 kilobits per unit of time, and the average size of the packets is 1 kilobits. For the dataset we consider the following parameters:

- **Topology:** We explore 3 different network topologies: unidirectional ring, star and scale-free networks.
- **Network size:** We study networks from 3 to 15 nodes where all nodes are active transmitters and/or receivers.
- **Traffic Distributions:** We evaluate 4 different packet length distributions: deterministic (constant), uniform, binomial and poisson using a fixed average packet length. In all the cases the inter-arrival time is exponential.
- **Traffic intensity:** We explore different levels of saturations in the network by varying the traffic intensity. For this we transmit, among all pairs of nodes, a random value of traffic with maximum value (ρ_{max}).

- **Routing:** We explore four different routing configurations, which are detailed in section 4.4.3.

Overall we have generated 380 different data sets with the different configurations in order to assess the accuracy of the neural network. Each data set consist on 10,000 different simulation with random traffic matrices between 0 and ρ_{max}.

4.3.3 Regressors

The data set is used to train different regressors: Deep Neural Networks (DNN) (using one and two hidden layers) and polynomial regression. For the DNN we explore the following hyper-parameters: number of hidden neurons, the activation function, the learning rate and the regularization parameter. For the case of the polynomial regressor we explore its degree. From all the explored models we choose the best one using the cross-validation technique, please note that we use an independent test-set to evaluate the accuracy of the model.

In terms of implementation we use the Tensorflow library (version 1.2.1) to implement the DNN models. After a manual tunning of the hyper-parameters, we used the following configuration:

- Activation function: Sigmoid
- Number of hidden layers: Equal to the number of input, i.e., the square of the number of nodes in the network
- Maximum training epoch: 7,500,000
- Training Algorithm: Adam Optimizer
- Cost function: MSE with L2 regularization
- L2 regularization parameter: 0.00003

For the polynomial regularization, we used the LinearRegression tool of the sklearn (version 0.19). For degree higher than one, we omitted cross-product terms due to the high dimensionality of the data.

In the results we compute the accuracy of the models as "learning error", it is expressed as a percentage:

$$error = 100\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{d}_i - d_i)^2} \tag{4.1}$$

Where N is the number of samples averaged, including the size of the test set and the different pair of nodes.

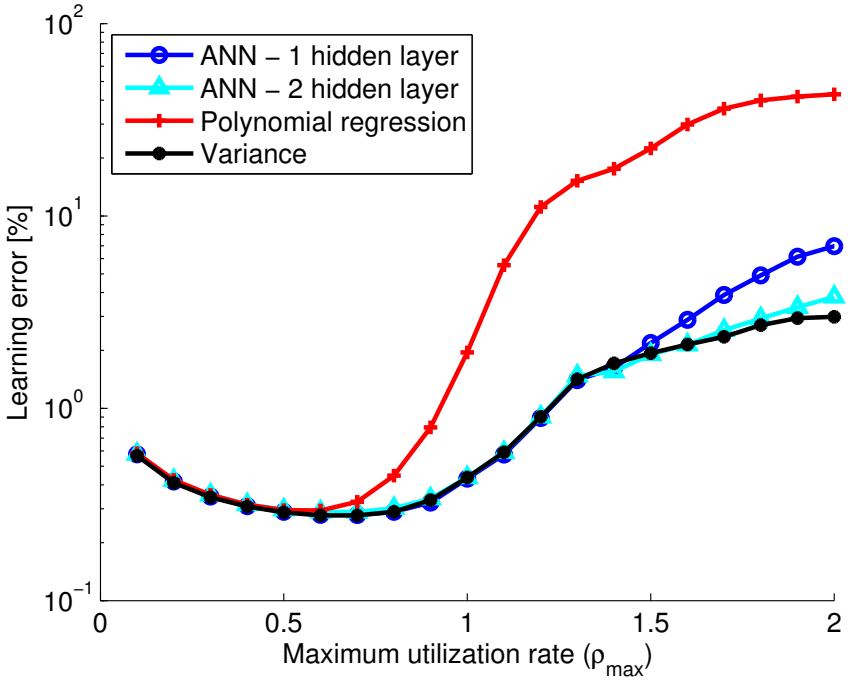

FIGURE 4.3: Learning error (log scale) as a function of the network saturation for a binomial traffic

4.4 Experimental Results

In this section we present the results obtained in five different experiments that cover different network and traffic scenarios. Please note that unless stated otherwise, we only show exemplifying figures since the other cases provide similar results

4.4.1 Traffic Characteristics and Intensity

First we focus on the accuracy of the neural network when estimating the delay of different traffic intensities and packet-size distributions. Figure 4.3 shows the accuracy of different regressors in a 10-node scale-free network with different traffic intensity and for the binomial packet size distribution.

Please note that the traffic intensity is expressed as (ρ_{max}). Each pair of node generates a uniformely distributed random bandwidth with maximum (ρ_{max}). As an example ($\rho_{max} = 2$) represents that, for each simulation, each node of the network, for each destination, generates a random traffic following the specified distribution at a random rate between 0 and the double of the link capacity divided by the number of destinations.

As figure 4.3 shows the neural networks exhibit an interesting behavior, first they perform remarkably well with an error below 5% in all the configurations. In order to better understand this error we also plot the variance of the data-set (black line

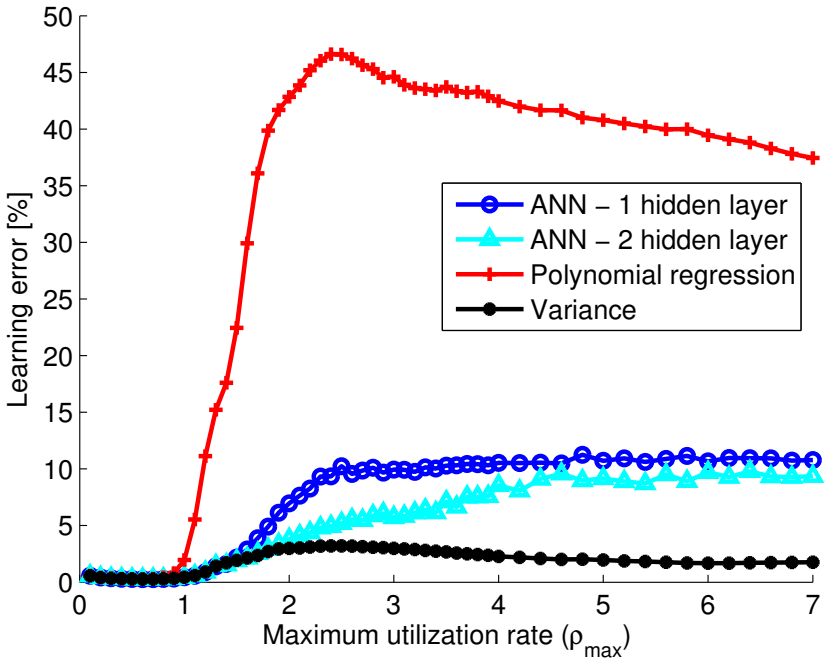

FIGURE 4.4: Learning error (lineal scale) as a function of the network saturation for a binomial traffic in more saturated networks

in the same figure). The variance is approximated by repeating 1,000 time each simulation, which is used to estimate the "real" delay, and using these samples in the variance approximation. The variance is also expressed as a percentage, as in (4.1).

Moreover, figure 4.4 shows the same experiment, in a lineal scale, and for more saturated traffic. It can also be observed than the performance of the polynomial regression when the network starts to saturate is really poor. However, we can also observe than the performance of the neural network models start deteriorate for more saturated traffic. This can be explained by two facts: 1) As the saturation increases, the model becomes more nonlinear and more difficult to learn, and 2) Our neural networks have been optimized in the region between one and two, and it start to diverge as we distance from this region.

Understanding the Variance of the training-set: As the figure shows the data-sets contains variance, this is the result of the measurement process. The per-packet delay measured in each simulation follows a certain distribution that has a variance. Then the mean end-to-end delay is estimated during a finite time: 16k time units in our experiments. As a result of using a finite number of samples, the variance results in an irreducible error when training the regressors. This error can only be reduced by increasing the number of samples.

Averaging time: In order to exemplify this effect we plot in figure 4.5 the influence of the averaging time used in the simulation (equivalent to the amount of packets) with the learning error. For this experiment we use a 10-node scale-free topology with a

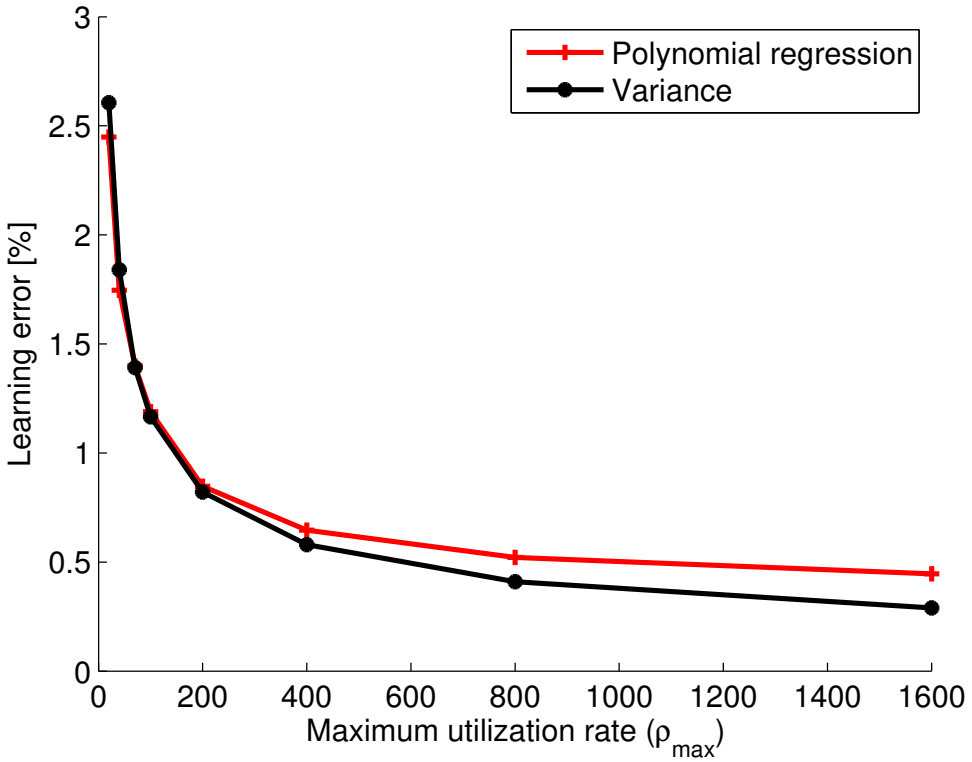

FIGURE 4.5: Learning error as a function of the averaging time

$\rho_{max} = 0.8$. As we expect the figure shows a sharp decrease of the learning error due to a reduction of the variance of the data. It is worth noting that this does not hold true when self-similar traffic is present since they have infinite variance. The main takeway from this is that, the measurement error of the data-set is propagated as learning error by the regressors.

Variance vs. Traffic Intensity: We have established that the data-set contains a variance as a result of the measurement process, but figure 4.3 also shows that the variance depends on the traffic intensity, *why?*. This variance in the delay of the packets is caused by two factors: (i) different transmissions times of the packets because of have different sizes and (ii) different queuing times. As the network becomes more saturated ($\rho_{max} > 1$) packets suffer different queing delays and as a result, the delay has larger variance. Once the network starts to approach full saturation ($\rho_{max} > 2$) a high percentage of packets find the queues full and as such, the resulting delay is more constant and the variance lower.

Other traffic distributions:

Figure 4.6, 4.7 and 4.8 shows the accuracy of different regressors in the same 10-node scale-free network with different traffic intensity and for other packet size distribution: a Poisson distribution, a deterministic distribution and constant packet lengths.

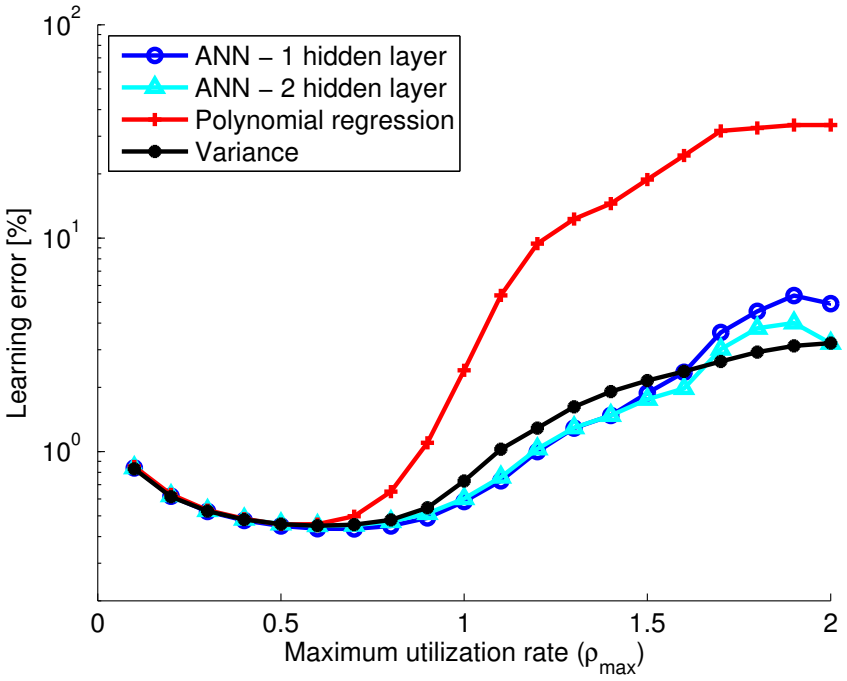

FIGURE 4.6: Learning error (log scale) as a function of the network saturation for a Poisson traffic

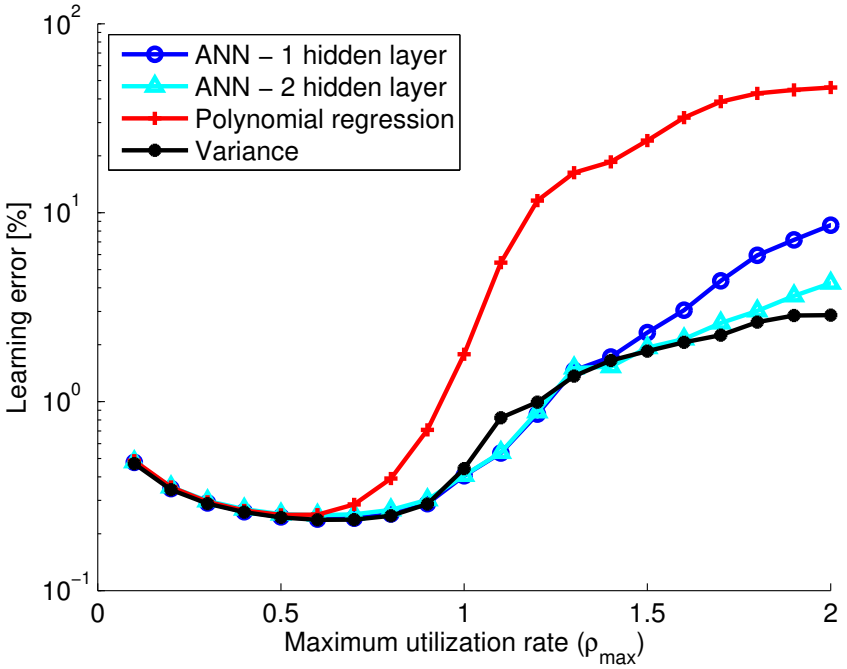

FIGURE 4.7: Learning error (log scale) as a function of the network saturation for a uniform traffic

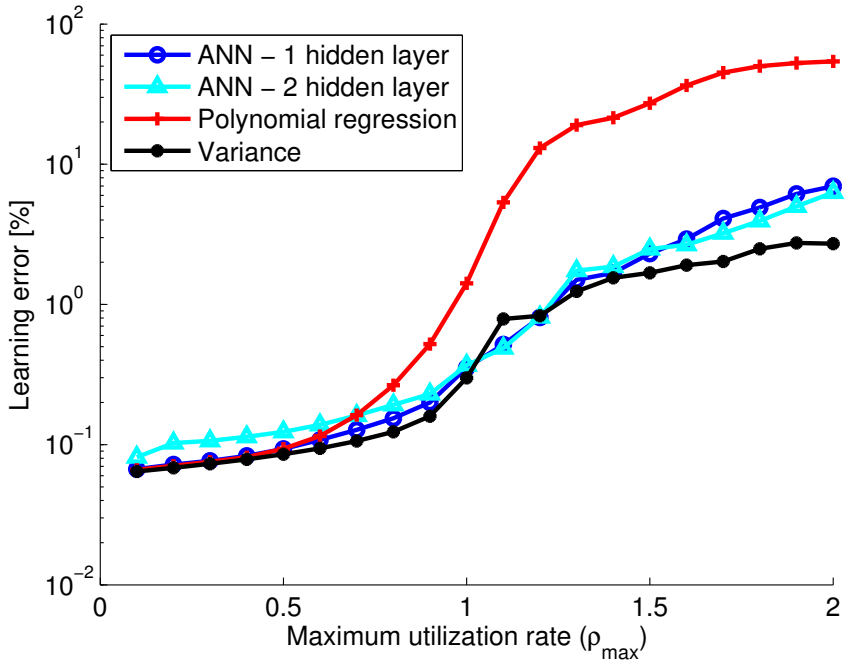

FIGURE 4.8: Learning error (log scale) as a function of the network saturation for a deterministic traffic

Figures 4.6 and 4.7, which represent the Poisson and uniformly packet length distribution, present a similar behavior than the binomial case. Note that in some points, it can be observed a learning error lower than the variance. This is because the variance is approximated, and in some case it may be overestimated. In both cases, the 2 hidden layers network presents always a better performance.

Finally, the results obtained using a constant packet length, presented in Figure 4.8 are little different. The behavior when the network is not saturated is different. The reason is that, in this case, the variance is only caused by the queuing time, whereas in the other cases it is caused also by the different size of each packet. The other difference is that, a priori, it can be observed a lower performance of the 2 layers network. This may be explained by the fact that the function learned in the deterministic case is more steep, which

Summary: With this we have established that:

- The training set contains variance that is the result of using a finite number of packets to measure the delay.
- This variance results in an irreducible error when training regressors
- The variance of the delay depends mainly on the queuing time.

With this we have a better understanding of figure 4.3. The neural networks provide accurate estimates of the delay and deeper networks (with 2 layers) can result

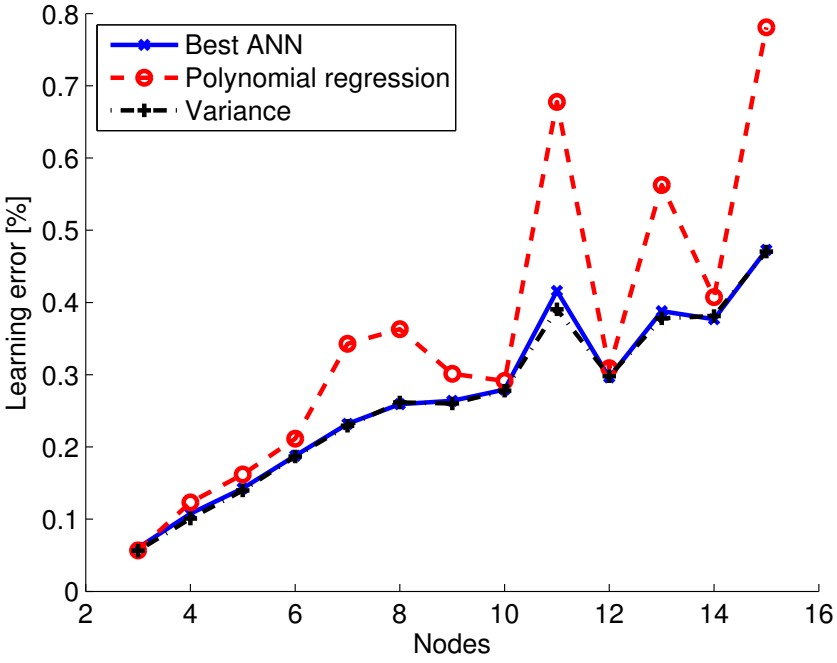

FIGURE 4.9: Scale-free topology

in a negligible error, almost matching the irreducible error resulting from the variance. The fact that deeper networks perform better suggests that the delay model is enough complex to justify the use of DNN, and that with deeper networks, this error can be reduced.

Figure 4.3 also shows that the polynomial regressor does not perform well, only in the presence of low intensity traffic, when the queues are practically always empty, they are able to estimate correctly the delay.

4.4.2 Topologies and Network Size

In this section we explore the accuracy of the neural network when estimating the delay with different network topologies and sizes.

Figures 4.10 and 4.9 show the accuracy of the regressors when estimating the delay in a ring and scale-free topology with different sizes, ranging from 3 to 15 nodes. In this scenario the traffic intensity is set to ($\rho_{max} > 0.6$). As the figure shows both regressors (polynomial and neural network) are able to produce accurate estimates with learning errors that match the variance. The main reason for this is that, with this traffic intensity and topologies, traffic does not suffer severe queuing. This can also be seen in the low variance of both scenarios. The exception is in some of the scale-free networks, in which some link may be saturated, which lowers the performance of the polynomial regression.

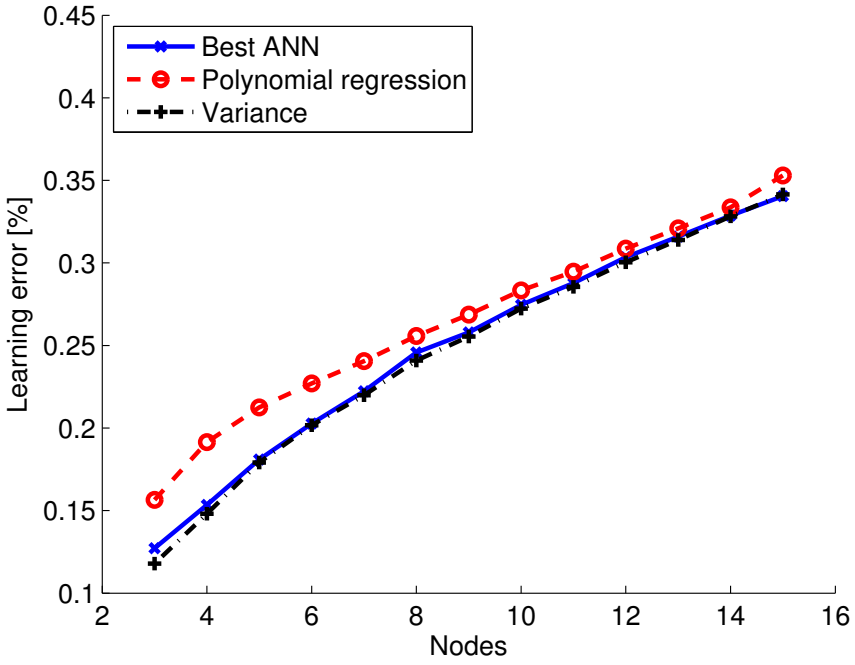

FIGURE 4.10: Star topology

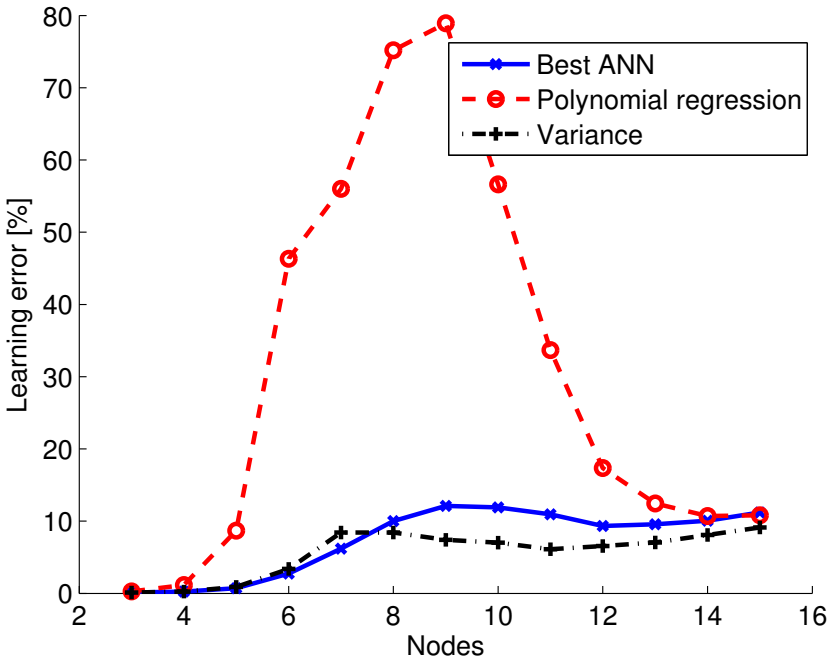

FIGURE 4.11: Ring topology

TABLE 4.1: Learning error using different routing

	NN	Poly	Variance
SP1	0.686%	0.722%	0.694%
SP2	0.683%	0.689%	0.674%
MAN	0.707%	0.703%	0.696%
POOR	2.241%	6.751%	2.314%

Figure 4.11 shows the accuracy for a ring topology with various network sizes, again ranging form 3 to 15. As in the previous case the traffic intensity is set to ($\rho_{max} > 0.6$). As the figure shows in this case the polynomial regressors performs poorly with high learning errors, on the other side the neural network shows an excellent accuracy, again matching the variance. The reason behind this is that the ring topology is easily saturated under different configurations, this results in higher variance and more complex functions to characterize.

Both plots shows that, for the considered use-cases, the topology or the size has no effect on the accuracy of the neural network estimates. The only impact of such fundamental network characteristics is that depending on the specific topology this may increase the saturation, which results in queuing and variance.

4.4.3 Routing

Finally and in the last set of experiments we explore the effect of the routing upon the learning capabilities. Table 4.1 shows the learning error when using four different routing policies in a scale-free network with 15 nodes and a traffic intensity of $\rho_{max} = 0.5$. For this we consider four different routing policies: 'SP1' and 'SP2' correspond to two different routing policies that follows the shortest-path approach, 'MAN' corresponds to manual designed routing policy, that may not follow the shortest path approach, but tries to load balance the use of the links. Finally, the 'POOR' configuration corresponds to a deliberately poor performance configuration, in which a few links are saturated.

In this case we observer that, in the three first scenarios, again both regressors perform well and that as in the previous cases they match the variance. The reason is that the traffic intensity is low and as such, the variance is also low. Please note how the 'POOR' routing policy performs worse since bottlenecks traffic through a few links that produces queuing and increases the variance. In such case the neural netowrk is able to match the variance while the polynomial regressors performs poorly.

From this experiment we establish that the routing policy has no effect on the accuracy of the estimates except when it causes queuing, which results in variance and higher errors.

4.5 Discussion

In this section we discuss our experimental results in order to have a better understanding of the use of neural networks for computer network modeling. Given the high computational cost of the experiments depicted in this chapter our results are limited to networks of 15 nodes, however we can extract several important lessons:

Neural networks can accurately model the average end-to-end delay as a function of the input traffic matrix for the considered scenarios: In all the cases the neural networks produced excellent performance. The main reason behind this is that, for the routing and forwarding mechanisms considered in the experiments, networks are deterministic complex systems with memory and can be learned with negligible error. As a result, neural networks should be considered as a relevant tool in the field of network modeling.

The main source of error of the neural networks are the variance of the data-set: The per-packet delay is a random process and estimating its mean with a finite amount of samples results in an error that propagates to the learning error of the regressors. This error can be reduced by using more samples, but longer measurement times may result in losing the stationary condition of the measure. Self-similar traffic also presents important challenges to reduce the measurement error.

This error is proportional to the variance of the delay process which in turn depend on the amount of queuing suffered by the packets. It is well-known that queuing depends on multiple factors: traffic intensity, routing, etc. This holds true not only for the delay but for any random process related to networks.

With this, we conclude that the main source of error for neural network modeling is the inherent error of estimating a random process with finite samples. For the particular case of the delay this is related to queuing. We have found that fitting error is negligible and we have not identified any other sources of error in our experiments. In addition, we have not found any inherent impact of fundamental network characteristics (size, topology, routing and traffic characteristics) in the accuracy of the neural networks, except when they result in additional queuing.

High traffic intensity requires deeper neural networks: Polynomial regressors have resulted in good performance except when the traffic intensity is high ($\rho_{max} > 1$). We have found similar results with neural networks that use one single layer. Therefore, networks that operate close to saturation require more sophisticated regressors. This

suggests that saturated networks have more complex functions that require deeper neural networks.

4.6 Conclusions

The main conclusion of this work is that the average end-to-end delay in communication networks can be accurately modeled using neural networks. We have found that neural networks perform remarkably well with a neglibigle error which is close to the irreductible measurement error.

The ultimate objective of this work is to lay the foundations for applying novel ML techniques in the field of network modeling. Future work includes applying these techniques in real scenarios with real traffic and protocols and develop ways to represent network-data (feature engineering).

Chapter 5

Improving the traffic forecasting with ML

5.1 Introduction

Within the context presented in this thesis, the traffic forecasting field plays an important role for the applications which are enabled by the KDN paradigm. Being able to estimate the future traffic of the network helps scaling up or down the software infrastructure to accommodate the load while providing efficient use of the resources.

Traffic forecasting is a well-established field, with a large existing literature proposing different models. Typically such models are based on time series mechanisms (ARMA and ARIMA) and work well with linear dependencies. Since traffic forecasting is not relevant only to computer network but to many other fields, considerable research efforts have been devoted to this topic in other fields, a notable example is in logistics (e.g., [37, 38]).

In this chapter we propose a methodology for traffic forecasting based on the combination of ARIMA and neural networks using external information (e.g, weather). Specifically, we use a traditional ARIMA model and we account for the outliers. Such outliers cannot be fitted by ARIMA because they typically have non-linear dependencies. However, such outliers are critical to the infrastructure, since these are the values that can overload the infrastructure or make an inefficient use of the resources. For this, we propose the use of a neural network to efficiently predict them. To train this neural network we use external information (weather, popular events, holidays, etc). The main hypothesis is that the traffic of the network depends on the behavior of its end-users, which in turn depends on external factors. In order to validate the proposed methodology we assess its performance using real-world data from a 30.000 users campus network, specifically the bandwidth of the egress Internet link averaged per day. Our results shows that our proposed methodology improves a traditional ARIMA model and that performs remarkably well when predicting outlier values.

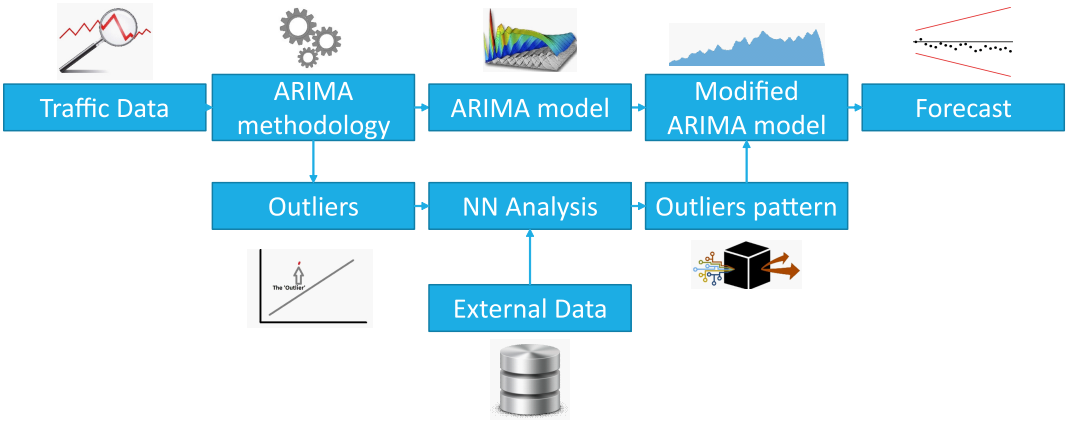

FIGURE 5.1: Methodology scheme

5.2 Methodology

In this section we describe the proposed methodology to combine time series techniques with Neural Networks to improve traffic forecasting, specifically to predict the bandwidth of an egress Internet link that aggregates traffic from a population of end-users. The Neural Network is used to improve the estimation done by the time series techniques by predicting the outliers that incorporate non-linear dependencies and that cannot be fitted by the time series techniques. The neural network is trained with external information to predict such outliers.

5.2.1 Overview

The methodology presented in this chapter is based on ARIMA modeling and forecasting, adding a non-linear section based on Neural Networks for outlier detection. An overview is shown in figure 5.1.

The first step to build the model is to obtain the *traffic data*. This data consists on a time series of the network aggregated traffic. The next step is to model the Internet traffic bandwidth data using an *ARIMA model* together with the Box-Jenkins methodology [54]. After the first modeling, we extract the *outliers*, which represent the traffic data that cannot be fitted using the *ARIMA model*. These outliers are processed with *neural networks* to extract their correlation with *external data*. The last step is to incorporate the information predicted by the neural networks, the *outliers pattern*, into the *modified ARIMA model*. Finally, the resulting hybrid ARIMA-ANN model is used to *forecast* the network traffic bandwidth taking into account historic traffic and external events that influence the behavior of this traffic. In what follows we describe each step in detail.

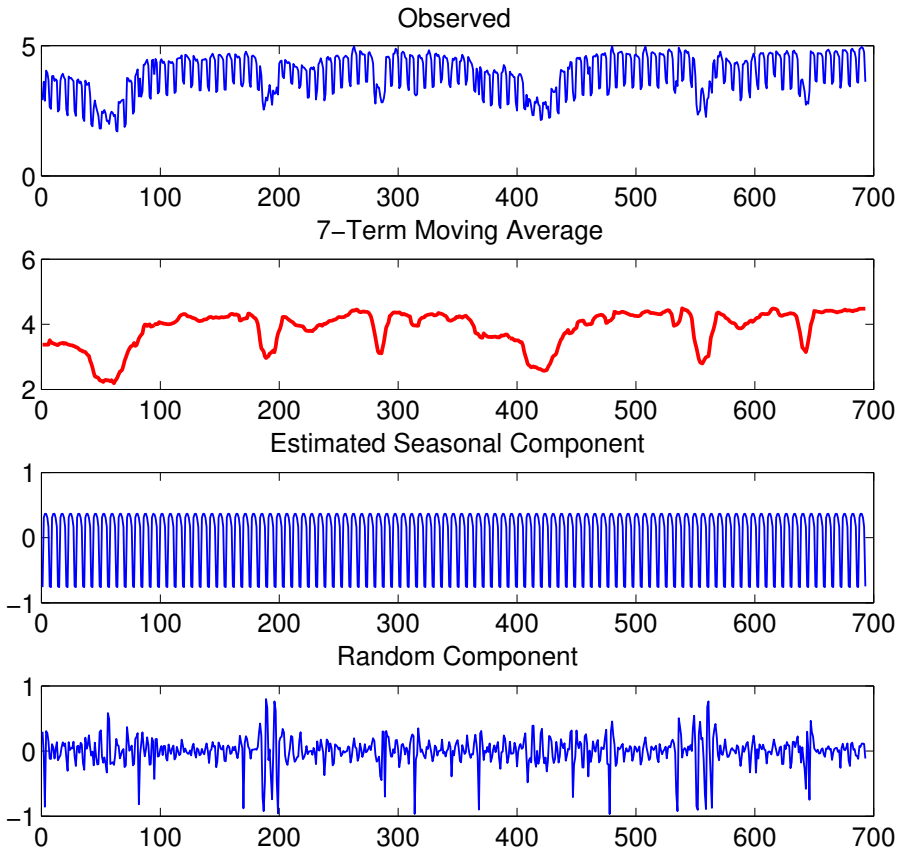

FIGURE 5.2: Time Series Decomposition (X axis: Days, Y axis: Bandwidth)

5.2.2 Time Series

The objective of the time series modeling is to predict the bandwidth of an Internet link only as a function of temporal information. For this we use the well-known ARIMA model. First we normalize the bandwidth data to ease its modeling and reduce its standard deviation. The second step is to determine if the traffic data that we are considering shows a significant volume growth over time. This growth is a symptom of a progressive increment of the variance, and is analyzed using time-dependent variances. In case traffic grows over time, the series is set to a logarithmic scale in order to minimize the increase of the variance without affecting its modeling. Finally, following Box-Jenkins methodology restrictions, the series needs to be stationary and of mean value equal to zero. These can be achieved using differentiation techniques.The goal is to get the stationary series with the lowest mean value ($m \approx 0$) and the lowest variance.

The second part of the modeling is to find the ARIMA coefficients to produce the ARIMA(p,d,q)(P,D,Q)s model that best fits the series. For this, the first step is to

determine the seasonality of the series. This is accomplished by observing the series plot and elaborating the series decomposition (see figure 5.2). The study of the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PCF) allows us to determine the degree of the ARIMA model and Seasonal ARIMA model [55]. The degree of the model can be detected by the Autocorrelation Function lags that overpass a threshold located at $\pm 2/\sqrt{N}$ in simple MA models. The degree of an AR model can be determined by the waveform of the autocorrelation function and the higher value lags of the Partial Autocorrelation Function. The components of the seasonal ARIMA model follow the same rules as the non-seasonal ones, applying them on multiples of S lags (being S the period of the time series). The Integrated Model (I) model degree marks the differentiation that lower the variance of the series the most and cannot generate a mathematical model of a time series by itself. It is applied to correct the stationarity of the series.

After the ARIMA model fits the time series, the residues provide information about the outliers present in the data. There are three main types of outliers: Additive Outliers (AO), Transitory Changes (TC), and Level Shifts (LS).However, in this study, only AO and TC outliers are considered, since LS outliers are caused by global changes in behavior of the time series and they usually imply a remodeling of the series.

The process required to extract the outliers from a time series is based on transforming the ARIMA pattern to an infinite AR one and comparing the residues between the model and the time series with the effect that the different type of outliers have on the time series.

For this we need to check for unusual high values in the residues, which indicates an outlier, and then find the corresponding outlier type that best fits them. The ARIMA model and the characteristics of the outlier determines its impact on the series. To obtain the final model, we subtract these outlier from the time series to increase the forecast precision. The next subsection exemplifies this process in a MA(1) to AR(∞) conversion.

Example of Outliers Detection (MA(1) to AR(∞))

In what follows we exemplify the outlier extraction process:

$$y_t = \epsilon_t - \theta\epsilon_{t-1} = \epsilon_t - \theta L\epsilon_t \tag{5.1}$$

Equation (5.1) defines the expression for a MA(1) model with mean value 0, which can be rewritten as a function of L, the delay of the samples.

$$\epsilon_t = \frac{y_t}{1-\theta L} \rightarrow \sum_{k=0}^{\infty} ar^k = \frac{a}{1-r} \tag{5.2}$$

The objective is to express y_t as an infinite sum dependent of y previous values from y_{t-1} to y_0. To accomplish this, we isolate ϵ_t from (5.1), and by using geometric series formula for $|k| < 1$ (see equation (5.2)) we obtain (5.3).

$$\begin{aligned} \epsilon_t = \sum_{k=0}^{\infty} y_t(\theta L)^k = y_t + y_t\theta L + y_t(\theta L)^2 + \cdots = \\ = y_t + y_{t-1}\theta + y_{t-2}\theta^2 + \cdots \end{aligned} \tag{5.3}$$

Finally, in (5.4) we express y_t as ϵ_t minus the infinite sum of its previous values. This expression determines the AR model that allows for outliers detection in this time series.

$$z_t = y_t = -\sum_{k=1}^{\infty} y_t(\theta L)^k + \epsilon_t = \epsilon_t - y_t - y_{t-1}\theta - y_{t-2}\theta^2 - \cdots \tag{5.4}$$

5.2.3 Neural Networks

The objective of the Neural Network (NN) module is to find correlations between the outliers detected in the time series and external data. The external dataset needs to contain temporal dependent features with possible impact on the network traffic. External data features are preprocessed to represent each value with a binary or ternary variable. The NN module has three functionalities: (i) detect when an outlier will appear in the future prediction, (ii) classify it on Additive Outliers (AO) or Transitory Changes (TC), and (iii) quantify its impact on the ARIMA model. To accomplish these objectives with precision and good performance, this module is structured with three levels of NN models that work in series, acting as data filters, as shown in figure 5.3. The first NN is a binary neural network that classifies the data as possible outliers or regular data. Only the set of values corresponding to possible outliers continues in this process to the second NN. This NN is a classifier that groups the samples in TC outliers, AO outliers or discards them. The last level of the NN module is composed by two NN that calculate the impact value of the different outliers classes. This last step also acts as a filter by applying zero to the non-outliers.

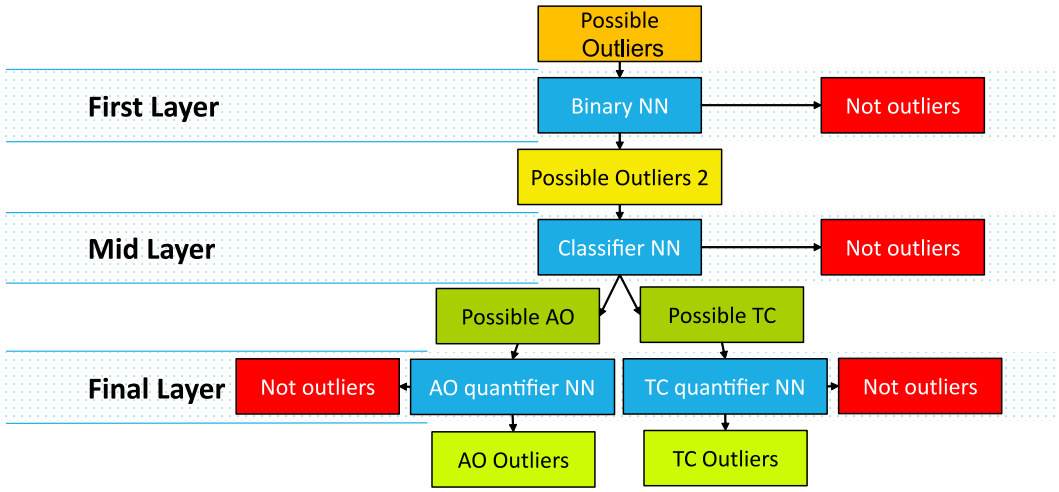

FIGURE 5.3: Neural Network Structure with three levels

5.2.4 Incorporating the Neural Network to the ARIMA model

The final step of the proposed methodology consists on incorporating the Neural Network (NN) models which predicts the outliers by using external data to the ARIMA model. For this, we need to subtract the original outliers from the time series before applying the ARIMA model to avoid the distortion caused by the outliers. This method increases the prediction accuracy and provides more precise forecasts when no outliers are present within the prediction range. By combining the prediction of the time series model with the NN module, we can prevent the error caused by any deviation that could be originated by external events. The impact of outliers, calculated by the NN model, is applied to the time series data once the forecast is made.

5.3 Experimental Evaluation

In this section we assess the validity of the proposed methodology proposed in section 5.2 experimentally. For this we compare the accuracy of a standard ARIMA model with our hybrid ARIMA+NN augmented with external information using a real-world dataset of a campus network.

5.3.1 Dataset

The dataset used for experimentation contains over 700 samples (2 years from 20/06/2014 to 18/05/2016) of the averaged daily bandwidth of the egress Internet link of a campus network with over 30.000 users. In order to correlate with external information we use the following features: precipitations and humidity, holidays, relevant

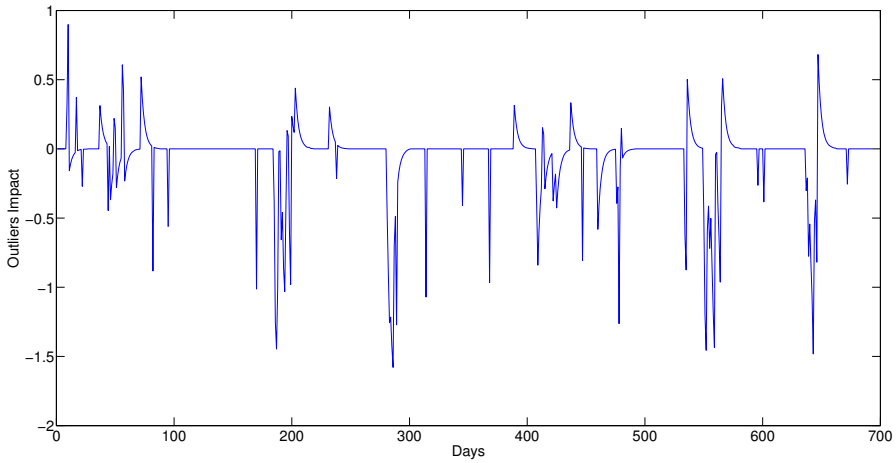

FIGURE 5.4: Outliers impact on the series

sportive and festive events hold both in campus and in the city, exam schedule, etc. The entire data-set is public and can be found at [12].

The time series samples are first normalized by 10^6, set into logarithmic scale and set stationary with a multiple differentiation of the samples at 1 and 7 lags (degree of the I model). The ARIMA model that best fits the series is an ARIMA(0,1,2) model with ARIMA(0,1,2)7 seasonal components. It is implemented with Matlab®, using the library Econometrics Toolbox [1]. This series presents strong outliers that can be appreciated in figure 5.4. In total 79 outliers are detected: 55 AO and 24 TC.

5.3.2 Neural Network

The Neural Network (NN) structure includes 3 different levels for outlier prediction (section 5.2.3) and in order to obtain each level we evaluate different NN configurations, this is implemented using Matlab®, making use of the Neural Network Toolbox library [2]. The NN structure is trained with 520 samples of the time series ($\approx 75\%$ of the total). In what follows we describe the process that we use to optimize the NN of each level.

The first level of the NN is the binary classifier, which needs to accomplish a high specificity rate in order not to filter out possible outliers. To find the optimal NN, each set of tested parameters has been trained 100 times to avoid anomalies. The data is distributed in training, validation and testing sets with the following percentages 80%, 10% and 10% respectively. The weight for the false negatives penalization while training is increased from 5% to 20%. The number of hidden layers

[1] MATLAB and Econometrics Toolbox, The MathWorks, Inc., Natick, Massachusetts, United States
[2] NMATLAB and Neural Network Toolbox, The MathWorks, Inc., Natick, Massachusetts, United States

is set in multiples of 10 from 50 to 100. The selected training functions are Conjugate Gradient [56], Levenberg-Marquadt [57] and Gradient Descent with Momentum Backpropagation [58]. The best NN obtained in this experiment is a Neural Binary Classifier with 100 hidden neuron layers, Gradient Descend with Momentum Backpropagation as training function and a penalization increase of 5%. This NN is able to discard more than half of the samples as non outliers with 0 false negatives.

The NN of the second level needs to be more restrictive and provide the least Transitory Changes (TC) and Additive Outliers (AO) false negatives. Following the same methodology as in the previous level the penalization increase tested for false negatives goes from 5% to 20%, the number of hidden layers ranges form 50 to 100 and the functions used for training are Levenberg-Marquadt and Gradient Descent with Momentum Backpropagation. The NN that achieves the best performance in this experiment is a NN Classifier of 90 hidden layers, Gradient Descend with momentum as training function and penalization of 5% more on false negatives. The performance of this network is similar to the previous one, only 2 of all the Additive Outliers are missed. After this step, the 86.35% of the non outlier samples is discarded.

The final level uses two NNs and aims to prioritize the correct weighting of AO and TC outliers in its impact on the time series. In this case, the parameters to test are the training function, that is chosen between Levenberg-Marquadt and Gradient Descent with Momentum Backpropagation, and the number of hidden layers, from 50 to 120. For both AO and TC outliers, the NN that achieves the best performance has 100 hidden neurons and uses Levenberg-Marquadt as training function. The output of the NN structure has a relative MSE of 1.6708 from the real outliers.

It is important to take into account that the performance of this structure could improve with a better data mining process for the external data and a deeper study of each layer of the module.

5.3.3 Results

In order to evaluate the accuracy of our novel methodology we first analyze its performance over a week, and we compare its performance with that of a standard ARIMA model. The standard ARIMA model does not incorporate the information of the external events, but we subtract the outliers before generating the model and we add them afterwards to increase its accuracy.

To exemplify the prediction differences between both models we present two examples of predicted samples affected by outliers. Figure 5.5 shows the data for 30 days and the forecasting for 7 days, both for the ARIMA model with Artificial Neural Networks (ARIMA-ANN) and the standard ARIMA model. In both cases the last days include an outlier and we can observe that the ARIMA-ANN model exhibits a

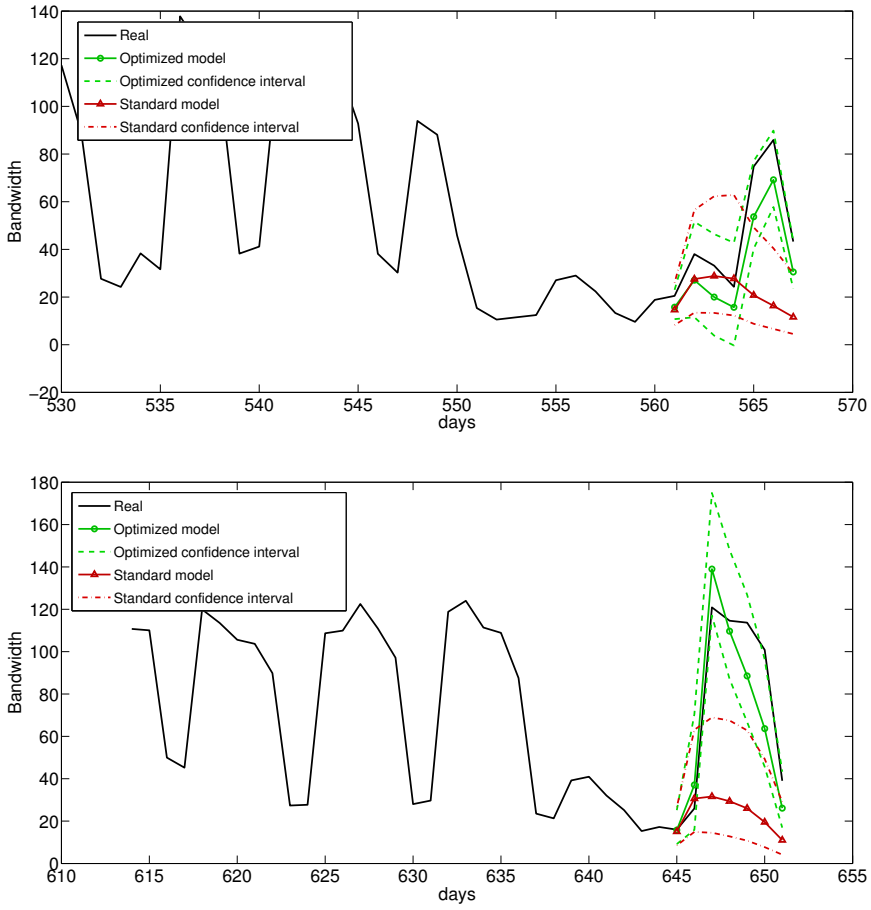

FIGURE 5.5: Forecast of 7 days using the standard ARIMA model and the modified one. Top: Forecast with an AO outlier. Bottom: Forecast with a TC outlier.

better performance. Particularly, figure 5.5 (top) shows a negative Additive Outlier (AO). The standard ARIMA model is not able to predict this deviation and the real data falls out of the confidence intervals (set to contain all prediction up to an accuracy of the 95%). The NN module predicts the AO with a considerably precision improvement. The confidence interval narrows in comparison with the standard model, which shows a higher prediction accuracy. Figure 5.5 (bottom) also shows a Transitory Change (TC) the week before of the prediction and an Additive Outlier at the end of the forecast week. The standard ARIMA model is able to adjust from the decrease of the previous week, but it cannot follow the shape of the real data. The ARIMA-ANN model can predict better the behavior of the traffic data.

The first full set of experiments aims at comparing the overall performance between the standard ARIMA model and the ARIMA model modified with NNs. It is based on the forecast of 25% of the samples of the time series, that is over 161 days of prediction. In this experiment the forecasting time is set to a week because longer estimations start deviating too much from real data and the comparison between both models lacks of relevance in such cases. The relative error ($\epsilon_r = \frac{\sum_{i=1}^{n} \epsilon_i}{\sum_{i=1}^{n} z_i}$), during the 161 days of forecasting is 28.32% for the ARIMA model and reduced to 17.58% for our model. For this same period of time, the Mean Squared Error ($MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Z}_i - Z_i)^2$) of the standard model is 980.076. In the combined model, the MSE decreases 63.93% to 353.486. To better understand the impact of this MSE we highlight that the time series mean value is 73.8478, once normalized by 10^6.

Finally, in the second set of experiments we compare the performance of both models taking *only* into account its accuracy when predicting the outliers. Typically network infrastructures are dimensioned for the standard load and, as a consequence, forecasting outliers is important since it may overload the infrastructure.

For these experiments we calculate the cumulative density function (CDF) of the normalized Mean Square Error (MSE) between the prediction and the real data for both models on the outliers not used for training the model (figure 5.6). An error equal to one refers to the maximum MSE of the standard ARIMA, which presents a higher error. As the CDF shows, our methodology works remarkably well, particularly for high error values which are significantly related with extreme outliers. This is caused by the better performance of the NN module for detecting high impact outliers. This experiment demonstrate that our model consistently outperforms the standard ARIMA being able to accurately predict outliers.

5.4 Discussion and Conclusions

In this chapter we have introduced a novel methodology for traffic forecasting, combining an ARIMA model with a neural network. The goal is that the neural network improves the prediction of outliers using external information (e.g., weather, sports

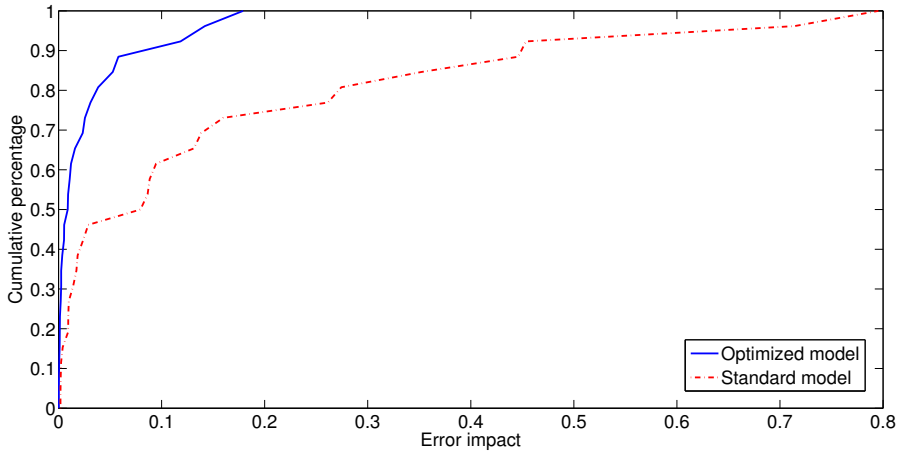

FIGURE 5.6: Prediction Error Cumulative Density Function

events, etc.). The methodology has been validated using real-world data from an egress Internet link of a campus network.

The main hypothesis behind this work is that the traffic depends on the users of the network, which in turn depend on external factors. This is a reasonable assumption for end-user networks, this is validated by our experimental results since we were able to identify and represent a set of features that were relevant for the users of the network under study.

As a consequence, in order to successfully apply this methodology to other networks, we need to identify the set of features that explains the behavior of the network under study. This may be difficult for networks that massively aggregate traffic, such as transit networks, since their population is very large and can be very heterogeneous. The same applies for networks that provide connectivity to datacenters with a wide variety of services.

Chapter 6

Complex network elements modeling

6.1 Introduction

Traditional computer networks are built from a large number of network elements (switches, routers, firewalls, load balancers...). The configuration of all these elements is a complex task and it makes the management of the network a really challenging task. Moreover, each network element may use different languages and tools to be configured, adding complexity to the network operator. Another important challenge in traditional networks is the "Internet ossification", which refers to the problematic caused by the infrastructure, since it is extremely difficult to evolve, either in therms of the physical infrastructure as well as the protocols operating in the network [22]. For all these reasons, novel approaches to computer networks are being developed, such as Software-Defined Networking (SDN).

SDN decouples the data plane and the control plane, in order to facilitate the management of the network through the abstraction of the physical elements. It also makes possible the idea of programmable networks, which simplify the management of the network. Moreover, it offers a centralized control point of the network, the SDN controller, which is connected to all the network elements and it is aware of the network configuration, topology and traffic. The SDN controller is the abstraction layer between the language of the network programmable elements and a higher-level language, but it is not responsible of the "intelligence" of the network; it is still requires software applications or the human intervention to manage network.

Within this context, we want to present our vision to simplify the management of the network and to achieve higher levels of optimization. We propose to build a network model by taking advantage of the centralization offered by the SDN paradigm. It can be built by using the centralized information to learn the behavior of the network. This network model can be used to predict different performance metrics, such as latency or energy consumption of the network in order to go towards an autonomous management of the network. State-of-the-art ML techniques are able to

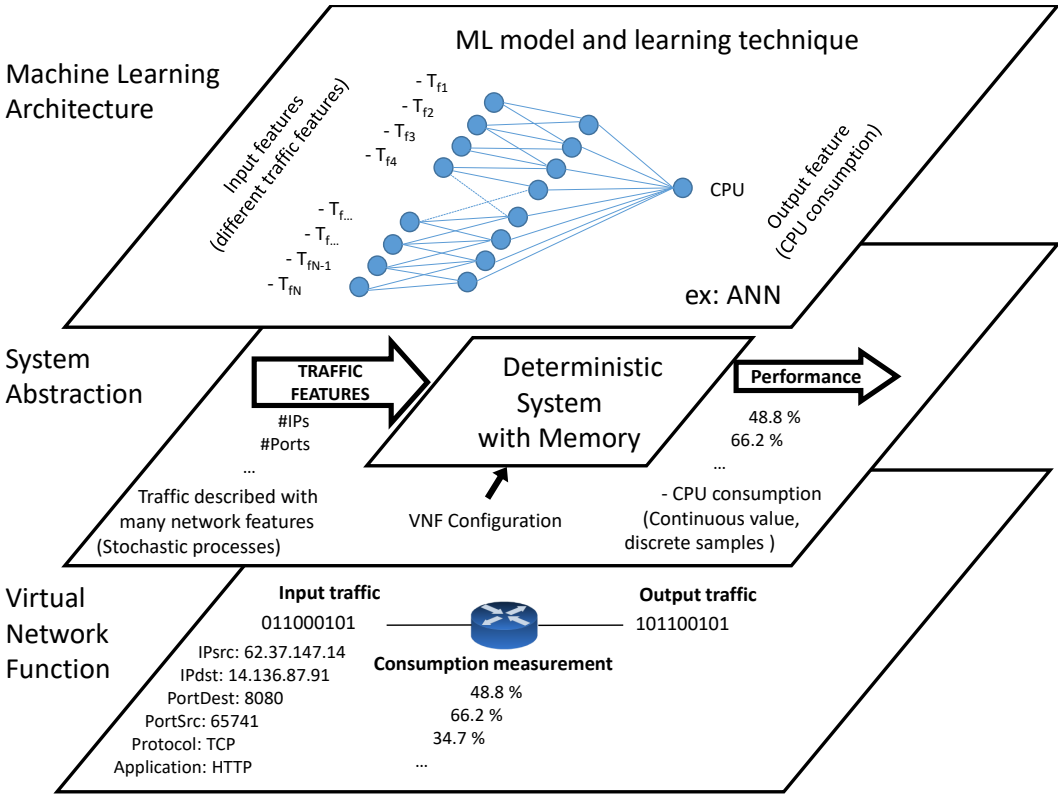

FIGURE 6.1: Graphic representation of the VNF performance modeling addressed in this chapter

learn difficult models to obtain good predictions. The fact of modeling the network behavior and being able to predict certain performance metrics can be extremely useful to increase the capabilities of SDN applications or to improve the performance of the network.

In this chapter, we explore the requisites, the applications and the viability of this vision. In Section 6.2, we present the problem we study and the questions we want to answer in this chapter. We present the methodology followed in the experiments performed in this chapter in Section 6.3. In Section 6.4, we present the experiments performed validate the hypothesis presented in the problem statements. Finally, we conclude the article in Section 6.5.

6.2 Problem Statement

In this section we describe the problem statement that we address in this chapter. We follow a similar approach that the followed in other chapters. Figure 6.1 summarizes the problem statement using three layers. The bottom layer represents the real-world VNF that we aim to model. The behavior of this VNF depends on its configuration, which in this work, we assume that it is constant.

The middle layer represents the system abstraction where the VNF is assumed as a black box, traffic ingresses the box and egresses it, altering the consumption of the VNF. In this scenario, we only have one source traffic, which is described my different features that describe its characteristics. In Section 6.3.2, these features are detailed. As in the network modeling scenario, the VNF can be abstracted as a deterministic complex system with memory, when random process such as physical errors are not taken into account. Additionally, its behavior, and consequently, its performance can be altered by changing its configuration.

Finally, the top layer represents the neural network that models the VNF performance. The neural network is able to produce estimates of CPU performance considering the input traffic, represented with the selected set of features. The VNF characteristics (exact function, configuration, ...) are hidden from the neural network, and hence the neural network is trained only for one particular configuration of the VNF.

The main questions that we aim to address in this chapter are:

- Can we train a neural network to produce accurate estimates of the CPU consumption of different VNF considering only the input traffic represented by features?
- Which is the impact of the features representing the traffic? Which ones are relevant? How many of them do we need?

6.3 Methodology

In this section we detail the methodology used to understand if a neural network can estimate the CPU consumption of a VNF as a function of the input traffic characteristics.

6.3.1 Overview

Figure 5.1 shows an overview of the methodology. In order to experimentally evaluate the accuracy of the neural network modes we use real traffic traces, which we reproduce in a controlled environment to be processed in the different VNFs. This traffic is continuously processed in the VNFs, in which we measure the average CPU consumption in time batches of 20 seconds. The traffic features of the traffic are also processed in the same 20 seconds batches. The data set is formed by these traffic features as the input features and the CPU consumption as the output features.

Once we have generated the data set we use them to train a a neural network, then we evaluate its accuracy using the cross-validation technique. We split the data set into three sets, the training set with 70% of the samples, the validation set with 15%

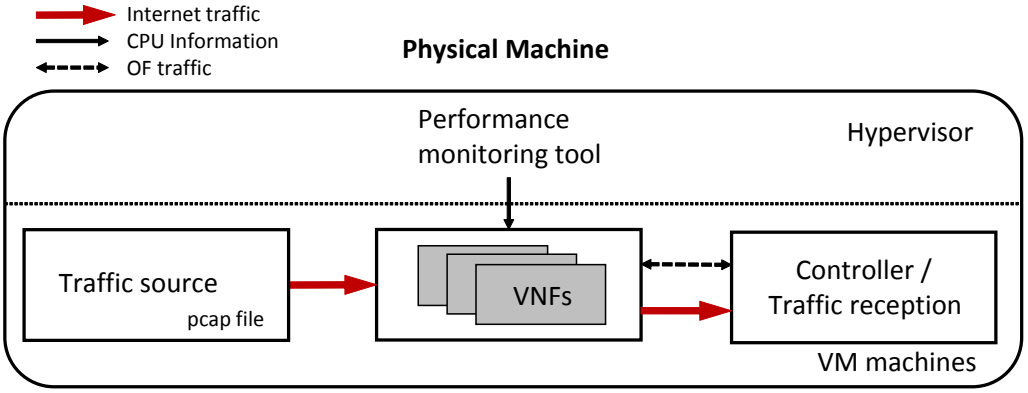

FIGURE 6.2: Scheme describing the configuration set up and the different VM used

of the samples and the test set with the remaining 15% of the samples. Finally the training set is used to optimize the ML model, the validation set is used to evaluate the model during the training phase and the test set is used to provide an independent evaluation of the mode . With this we compare the CPU consumption estimated by the regressors with the one measured in our experiment.

6.3.2 Traffic

We have tested the different VNF with real traffic, captured in an on-campus link [36]. Particularly, we have used two 5 min long traces of traffic, captured in two different moments of the day in a 10 Gbps campus interconnection link (both ingress and egress). Then, this traffic is reproduced in the test environment and sent to the VNF to measure the CPU consumption in this scenario.

Since the speed of the link is too high to reproduce the traffic, we have scaled it by dividing the rate per 10 and by 40 (night traffic and day traffic respectively), in order to obtain around 2,000 packets per second.

To describe the traffic in this particular experiment, we have used a set of 86 features which describe network, transport and application level attributes. You can find the complete set of features in table 6.1

6.3.3 VNFs

This experiment is implemented with real VMs implementing two different VNF: OVS [59] and SNORT [60]. Open vSwitch (OVS) is an open-source implementation of a virtual SDN multilayer switch, which is used nowadays in multiple hypervisors. SNORT is a network intrusion prevention system (NIPS) and network intrusion detection system (NIDS), which is able to analyze real-time traffic.

TABLE 6.1: Complete set of features used to describe the traffic

Batch number	Num. of different source ports in TCP packets	Average inter-arrival time between consecutive TCP flows
TimeStamp (in us) of the last packet	Num. of different destination ports in TCP packets	Std of the inter-arrival time between consecutive TCP flows
Number of packets	Num. of different source ports in UDP packets	Average inter-arrival time between consecutive UDP flows
Transmitted bytes	Num. of different source ports in TCP packets	Std of the inter-arrival time between consecutive UDP flows
Average inter-arrival time between consecutive packets	Num. of different pairs ipSrc-PortSrc	Average inter-arrival time between consecutive "Other" flows
Std of the inter-arrival time between consecutive packets	Num. of different pairs ipDst-PortDst	Std of the inter-arrival time between consecutive "Other"
Average length of the packets	Num. of SYN-TCP packets	Average length (in bytes) of a TCP flow
Standard deviation of the length of the packets	Num. of FIN-TCP packets	Standard deviation of the length of a TCP flow
Num. of different IP source addresses	Num. of RES-TCP packets	Average length (in bytes) of a UDP flow
Num. of different IP destination addresses	Num. of different flows	Std of the length of a UDP flow
Num. of different pairs IP source-destination addresses,Num. of different IP destination addresses	Num. of different TCP flows	Average length (in bytes) of a "Other" flow
Num. of IPv4 packets	Num. of different UDP flows	Std of the length of a "Other" flow
Num. of IPv6 packets	Num. of unidentified TCP or UDP packets	Num. of flows with only one packet
Num. of the ICMP packets over IPv4	Num. of "othersL4" flows	Num. of http packets, identified by the source or destination port (80, 8080, 8008, 443)
Num. of the ICMP packets over IPv6	Average number of packets in a TCP flow	Num. of http flows, identified by the source or destination port (80,8080, 8008, 443)
Num. of different IP source addresses mask 30..16 [15 different features values]	Std of the number of packets in a TCP flow	Num. of ssl packets, identified by the source or destination port (22)
Num. of different IP destination addresses mask 30..16 [15 different features values]	Average number of packets in a UDP flow	Num. of ssl flows, identified by the source or destination port (22)
Num. of TCP packets	Std of the number of packets in a UDP flow	Num. of smtp packets, identified by the source or destination port (25, 465, 587, 2525, 3535)
Num. of UDP packets	Average number of packets in "Other" flow	Num. of smtp flows, identified by the source or destination port (25, 465, 587, 2525, 3535)
Num. of other packets	Std of the number of packets in “Other” flow	

Two different configurations are used in the OVS. The first configuration is the common behavior of an SDN switch. It implements some rules which are able to forward most of the packets, but some of them needs to be sent to the controller. In this case, the controller installs a new rule in the OVS, which matches and forwards similar packets. This rules expires after 10 seconds, in order to keep limited the number of rules in the OVS. The second configuration of the OVS consist on implementing a large number of rules (around 300) that act as a firewall. These rules can forward, discard or modify the packets going through the OVS. The controller is not used in this configuration, but the complexity and the number of predefined rules is considerably increased. A single configuration is used in the SNORT VNF, the default configuration. This detects the most common attacks and undesirable traffic.

6.3.4 Set Up

All the components needed to complete the experiment are deployed in the same physical machine using different VM under the same hypervisor and interconnected with virtual network links from the hypervisor. Each VNF is installed in a different VM and the CPU consumption of the complete VM is measured by using the performance monitoring tool of the hypervisor, which gives the average CPU performance in 20 seconds batches. The traffic is generated in a second VM and sent through the virtual network to the VFN, which process it. A third VM is used as an SDN controller and to receive the traffic. The complete diagram is shown in Fig. 6.2.

The traffic features are calculated off-line, in the same 20 seconds batches that offers the monitoring tool of the hypervisor; and the CPU consumption is collected for all the traces. Once the data is collected, it is used to train the ML algorithm. The total number of training examples obtained with the all the traces is 1359 points.

This experiment consists in predicting the CPU consumption of different VNF only from the network traffic. The network topology is constant and we only explore two VNFs and three different configurations. The CPU consumption of a VNF depends on the own VNF (the network function that it executes and the implementation), the configuration of the VNF and the traffic load that it handles. Therefore, for a certain VNF implementation and for a known configuration, the CPU consumption depends only on the traffic.

6.3.5 Machine Learning training

The ML technique chosen is Artificial Neural Network (ANN), and we have used the MATLAB Neural Network Toolbox library [1]. Particularly, we use one hidden level with five neurons and one neuron in the output level. The MATLAB toolbox

[1]NMATLAB and Neural Network Toolbox, The MathWorks, Inc., Natick, Massachusetts, United States

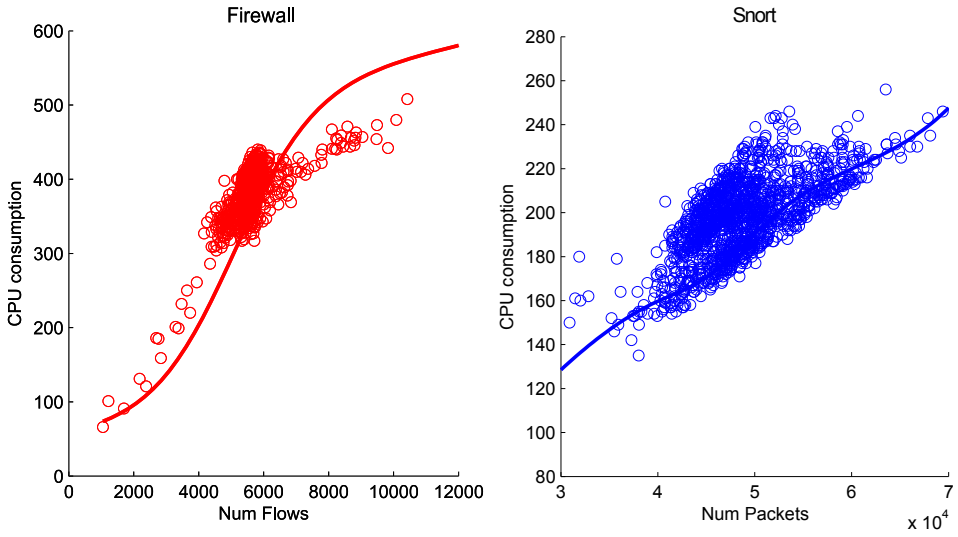

FIGURE 6.3: Model built using two different features for two different VNF (only the relevant feature is shown)

divides the training set randomly in three independent sets: training, validation and test, in order to optimize and evaluate the training and the training parameters. The Levenberg-Marquardt algorithm is used in the training process and the mean squared error is used as the error metric.

6.4 Experimental results

6.4.1 Results and discussion

The first step consists on observing the dependence of the CPU consumption as a function of different features for different VNFs. We choose two relevant features, the number of packets and the number of flows and we train the model only with these two features. Fig. 6.3 shows the observation points and the model built with these features. For the OVS configured with firewall rules, we observe that the model is built only from the number of packets (as shown in Fig. 6.3 left), and the number of packets is considered irrelevant (no CPU consumption independently of the number of packets, not shown in the figure). For the SNORT VNF, the opposite occurs, the CPU consumption is modeled using only the number of packets and not the number of flows (see Fig. 6.3 right). This is mainly because the SNORT has to inspect each packet individually, whereas the OVS configured with firewall rules has to decide what to do which each flow. However, as can be seen in the figure, the model build with these two features is not good enough and the error in the prediction is big. For this reason, a bigger set of features describing the traffic is needed.

When we train the model with all the 86 traffic features, the CPU consumption of these three configurations can be predicted with a small error. Fig. 6.4 represents

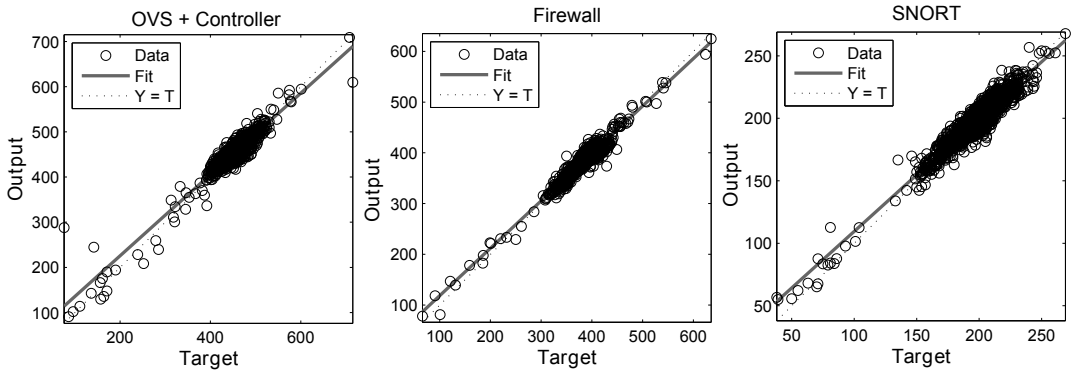

FIGURE 6.4: Predicted value vs actual value in the three different configurations

the predicted CPU consumption as a function of the real CPU consumption. The closer to the diagonal, the better the prediction is. In this figure, we observe that the prediction is good and all predicted values are close to the diagonal. In this case, it is not possible to represent the model as a function of the features, since they are correlated and a single feature by itself is not relevant.

Fig. 6.5 shows the CDF of the absolute value of error (deviation) in percentage. It can be observed that the CPU consumption can be predicted with less than 2 % of error in half of the observations, and less than 10 % of error in almost all the observations. Moreover, the relative error is similar, keeping in mind that the absolute CPU consumption of the three configurations is different. In addition, we can consider the mse (mean squared error) to confirm the validity of these models. A higher mse error, implies either that the average consumption is higher or that the prediction is worse, for this reason we also give the mean CPU consumption. The OVS using a controller presents a mse around 250, which corresponds to a std of $\approx$ 16 MHz (mean CPU $\approx$ 450). The OVS behaving as a firewall presents a mse around 180, which corresponds to a std of $\approx$ 13 MHz (mean CPU $\approx$ 375 MHz). Finally, the SNORT application presents a mse of 55, which corresponds to a std of $\approx$ 8 MHz (mean CPU $\approx$ 191 MHz).

During the training process, the ML algorithm learns and discover the relationship between the features and the CPU consumption. Some features may be more relevant, less relevant or completely irrelevant for some VNFs and have a different relevance for a different VNFs. Fig. 6.6 shows the average error (in %, normalized by the absolute cpu consumption) of applying the ML algorithm into single features. The error is sorted to facilitate the analysis of the graphic, thus, the points on the x-axis do not represent the same feature. We can observe than a small group of features can be used to predict the CPU consumption with a small error, the vast majority present a certain correlation but with a bigger error, and some of them cannot be used to predict the CPU consumption. Which features are important or det depend on the VNF and on the configuration. The ML technique is able to discover the relationship among the relevant features and with the CPU consumption and ignore the

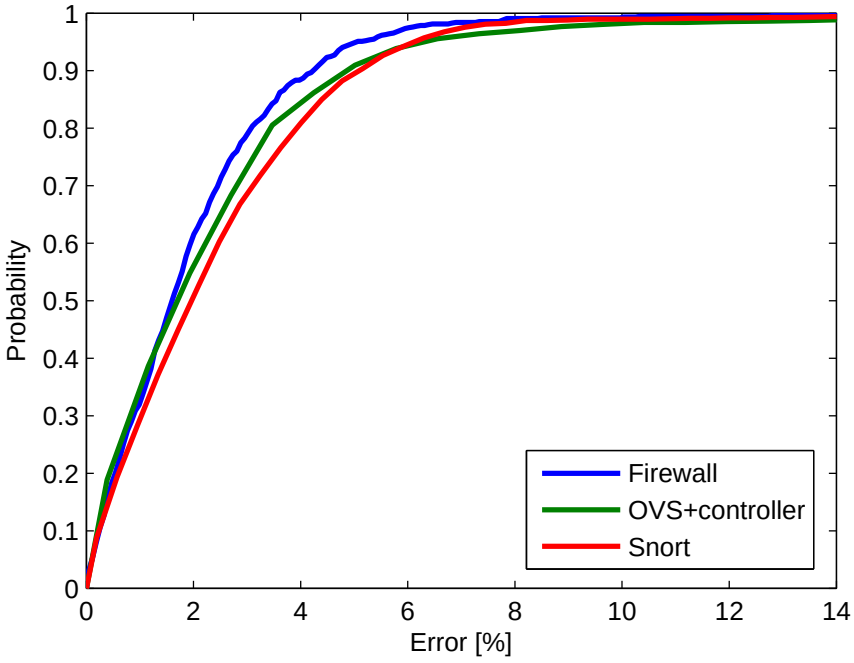

FIGURE 6.5: Cumulative distribution function of the error as a function of the percentual error

irrelevant ones. The case showed previously (Fig. 6.3) exemplifies this.

In this experiment, similar results can be obtained with simpler regression models, such as multiple linear regression (using higher degree polynomials if needed). However, the main advantages of ML is that can easily incorporate not linear behaviors in the same ML model. For example, the same ML model can be used to predict the CPU consumption of these three configurations of VNF, by only adding one feature indicating each configuration. We foresee that some of the features that describe the topology and the configuration of the network cannot be used in simpler lineal models.

We conclude that a ML-based model can be used to predict a performance metric, in this experiment the CPU consumption, in a certain network, in this experiment a single network element. With these results, we envision that the proposed network model can be built in more complex networks and to predict other performance metrics.

6.5 Summary and Concluding Remarks

In this chapter we have modeled the performance of different Virtual Network Functions (VNF), specifically we have focused on the CPU performance of the VNF observed by the hypervisor. We have modeled this behavior as a black-box, and only as a function of the incoming traffic. The main challenge here is how the accurately

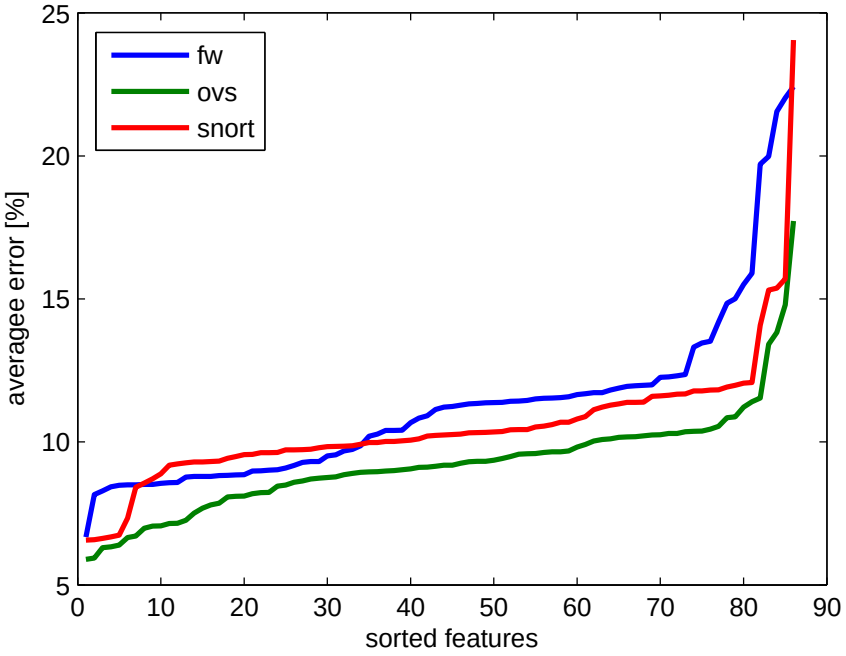

FIGURE 6.6: Sorted training error of single features

represent the traffic, which clearly determines the behavior of this functions. We used a big set of features representing different characteristics of the different network layers.

We have demonstrated that this behavior can be learned, that the function we are learning is neither trivial or linear, and that this function depends on different features for different functions. Future work includes how to represent the configuration of the VNF to be used in the learning process, to be able to use the same model for different configurations.

Chapter 7

Conclusions and future work

In this chapter, we conclude the work presented in this thesis and we present the future work and open research challenges on the deployment of the KDN paradigm.

7.1 Conclusions

In this thesis, we first introduced the concept of Knowledge-Defined Networking (KDN), a novel paradigm that combines Software-Defined Networking, Network Analytics and Machine Learning to ultimately provide automated network control. We also presented a set of use-cases that illustrate the feasibility and advantages of the proposed paradigm. KDN relies heavily on the use of ML techniques to learn network models, with the ultimate goal of providing automated network control. We also discussed some important challenges that need to be addressed before completely achieving this vision. We advocate that addressing such challenges requires a truly inter-disciplinary effort between the research fields of Artificial Intelligence, Network Science and Computer Networks.

Afterwards, we have studied the performance of two different models to characterize the delays in a network given the traffic load. We have demonstrated that simple neural network based estimators performs remarkably well. However, these estimators models the network as a black-box and do not provide insides of the behavior of the network. For comparison, we implemented a M/M/1-inspired estimator, which could provide more information of the network, but it has leaded to poor results.

Consequently, we have focused on the black-box network modeling. We verified that the average end-to-end delay in communication networks can be accurately modeled using artificial neural networks. Specifically, we have studied the modeling capabilities, under different traffic characteristics, and for different network topologies, network sizes and routing configurations. We have found that neural networks perform remarkably well with a negligible error which is close to the irreducible measurement error. The ultimate objective of this work is to lay the foundations for applying novel ML techniques in the field of network modeling.

In this thesis, we have also studied the application of ML techniques in other scenarios. We introduced a novel methodology for traffic forecasting, combining an ARIMA model with an artificial neural network. The goal is that the neural network improves the prediction of outliers using external information (e.g., weather, sports events, etc.). The methodology has been validated using real-world data from an egress Internet link of a campus network. The main hypothesis behind this work is that the traffic depends on the users of the network, which in turn depend on external factors. This is a reasonable assumption for end-user networks, this is validated by our experimental results since we were able to identify and represent a set of features that were relevant for the users of the network under study.

Finally, we have studied the application of the KDN architecture in a novel networking paradigm, the Network Function Virtualization (NFV). We have modeled the performance (specifically the CPU performance) of different Virtual Network Functions (VNF) as a black-box, and only as a function of the incoming traffic. The main challenge here is how the accurately represent the traffic, which clearly determines the behavior of this functions. We used a big set of features representing different characteristics of the different network layers.

The main conclusion of this thesis, is that the use of Machine Learning techniques in the networking field presents a promising future, enabling a new set of applications and simplifying the control and the management of current and future networks.

7.2 Future work

The work presented in this thesis has shown how ML techniques and the KDN architecture can be useful in different applications and scenarios. However, there is still work to be done and important research challenges that need to be addressed before use them.

7.2.1 KDN challenges

The novel KDN paradigm represents a new networking paradigm, but it presents many challenges introduced in Chapter 1, and summarized hereunder.

KDN enable new set of applications for ML and as such, requires either adapting existing *ML mechanisms* or developing new ones. A notable example graphs, which are used in networking to represent topologies, which determine the performance and features of a network. Moreover, the combination of modern ML techniques, such as Q-learning techniques, convolutional neural networks and other deep learning techniques, may be essential to make a step further in this area.

In contrast of traditional models and techniques, models produced by ML techniques *do not provide deterministic results* and are difficult to understand by humans. This also means that manual verification is usually impractical when using ML-derived models. Nevertheless, ML models work well when the training set is *representative* enough. Then, what is a *representative* training set in networking? This is an important research question that needs to be addressed. In some use-cases a training phase that tests the network under various representative configurations can be required. In this scenario, it is necessary to analyze the characteristics of such loads and configurations in order to address questions such as: does the normal traffic variability occurring in networks produce a representative training set? Does ML require testing the network under a set of configurations that may render it unusable?

Finally, we need a *standardized high-quality training datasets* to test, develop and benchmark new ML algorithms, since focusing on the dataset rather than on the algorithm may be a more straightforward approach. The publication of datasets is already a common practice in several popular ML application. In this thesis, we advocate that we need similar initiatives for the computer network AI field. For this reason, all datasets used in this thesis are public and can be found at [12]..

7.2.2 Network and traffic modeling challenges

In the experiments presented in this thesis, we have demonstrated that the end to end delay of relatively small networks can be modeled. However, an important challenge is the ML field is how to deal with the high dimensionality of the data needed to model a network. Indeed, to completely model the end-to-end delay of a network, we need to represent: 1) K features to model a flow, 2) F flows to be model among each pair of nodes, 3) N^2 pairs of nodes (in N represents the number of nodes).

These problem may be addressed in two different ways: 1) From a ML point of view, we need to study models able to deal with all these features. A possibility is to use the idea of the auto-encoders to reduce the dimensionality of the data. 2) From a feature engineering point of view, we need to study if we can represent all this information in a different way. A possibility is to study patrons among real traffic features which may suggest a small set of features to represent the traffic.

7.2.3 Traffic forecasting challenges

In this thesis, we have demonstrated that the use of external information can improve the detection of outliers when forecasting traffic. However, in order to successfully apply this methodology to other networks, we need to identify the set of external features that explains the behavior of the network under study. This may

be difficult for networks that massively aggregate traffic, such as transit networks, since their population is very large and can be very heterogeneous. The same applies for networks that provide connectivity to data-centers with a wide variety of services.

7.2.4 Complex network elements modeling challenges

Complex networks, with virtual elements, may be more complex to model, specially when these functions depends on specific traffic characteristics. It is an important challenge to verify is the same set of features needed to represent "conventional" networks is enough to model these "advanced" networks.

7.2.5 Topology and configuration challenges

In this thesis, we have focused on models for networks or elements with a constant topology or configuration. However, real networks or elements may change its topology or its configuration. Being able to capture and learn this changes in the models we presented is an important challenge. The main difficulty is how the represent these information as a ML-understandable features.

Appendix A

Network Simulator

In this appendix, we describe in detail the simulator used in most of the experiments performed in this thesis. The objective of the simulator is to obtain useful datasets to be used to test the Machine Learning (ML) capabilities under different conditions.

A.1 Simulator requirements

The function of the simulator is to simulate the basic behavior of a computer network and to compute the end-to-end delay among each pair of nodes with random traffic following specified statistical characteristics.

There are some network simulator available, such as *ns* or *mininet*. We have discarded them due to the computation speed. They are too complex for our needs, which increases considerably the simulation time. Note that we need to obtain, many different datasets, which lots of samples each one.

The main requirements are:

- **Fast:** We need to do many repetitions of each simulation, as well as many different simulations. Therefore, the execution time is a key requirement.
- **Realistic:** We need to obtain realistic results, which discards the fluid simulator approach and makes necessary the use of packets. The minimum that the simulator has to emulate are the queues and the routing through a defined topology.
- **Configurable:** We need to perform many different simulations, such as: modifying the topology, the traffic characteristics or the routing. This changes have to be easy to simulate.
- **Traffic generation:** The simulator has to generate random traffic following different statistical distribution.

- **Configurable output:** We need to obtain different results, such as the average end-to-end delay, the variance, the number of packets lost... It has to be easy to change the results generated by the simulator

A.2 Design

We decided to implement the simulator on top of Omnet++ simulation framework [**omnet**], which is a discrete-event simulation, primary built for network simulators. We did not use any existing framework to implement only the needed feature in order to speed up the simulation process.

We took advantage of the modularity offered by Omnet++ to implement different modules and to interconnect them.

The basic modules of the simulator are:

- TrafficController: Generate the distribution of traffic of a node for each different destination
- Application: Generates the traffic as a function of the characteristics received from the TrafficController
- Routing: Routes the packets only as a function of the destination according to the routing table read from a file.
- Balancer: Routes the packets to balance the usage of the links which are connected to the balancer.
- NodeQueue: Represents one port. Sends the packet when the channel is empty and queues the packet when not. The size of the queue is finite, so when the queue is full, the packet is discarded.
- Node: It represents a node in an overlay-underlay network, which can send and receive traffic. This implies that it routes the traffic to the different ports of the underlay network according to the overlay routing policy.
- Router In represents a node of the network that does not send and receive traffic. It only routes traffic according to its own routing table,
- Server In represents a node of the network that sends and receives traffic. It routes its own traffic and the and the incoming traffic according to its own routing table.
- Receiver In represents a node of the network does not generate traffic but it receives traffic. It routes its own traffic and the and the incoming traffic according to its own routing table.

A.3 Use of the simulator

We have perform many simulators to obtain all datasets used in this work. To do so, we use the cluster of our department.

All datasets used in this thesis are publicly available on www.knowledgedefinednetworking.net.

Bibliography

[1] David D. Clark et al. "A knowledge plane for the internet". In: *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (2003), pp. 3–10.

[2] Diego Kreutz et al. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76.

[3] Changhoon Kim et al. "In-band Network Telemetry via Programmable Dataplanes". In: *Industrial demo, ACM SIGCOMM*. 2015.

[4] Luis Miguel Rios and Nikolaos V. Sahinidis. "Derivative-free optimization: a review of algorithms and comparison of software implementations". In: *Journal of Global Optimization* 56.3 (2013), pp. 1247–1293.

[5] K.S. Narendra and K. Parthasarathy. "Identification and control of dynamical systems using neural networks". In: *IEEE Transactions on Neural Networks* 1.1 (1990), pp. 4–27.

[6] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[7] Michele Zorzi et al. "Cognition-Based Networks : a New Perspective on Network Optimization Using Learning and Distributed Intelligence". In: (2015), pp. 1–17.

[8] Michele Zorzi et al. "COBANETS : a new paradigm for cognitive communications systems". In: Icnc (2016), pp. 1–7.

[9] Danilo Giordano et al. "YouLighter: A Cognitive Approach to Unveil YouTube CDN and Changes". In: *IEEE Transactions on Cognitive Communications and Networking* 1.2 (2015), pp. 161–174.

[10] Hongzi Mao et al. "Resource Management with Deep Reinforcement Learning". In: *HotNets* (2016), pp. 50–56.

[11] N. Foster et al. "Languages for software-defined networks". In: *IEEE Communications Magazine* 51.2 (2013), pp. 128–134.

[12] Albert Cabellos and Albert Mestres. *KDN database (public datasets for ML)*. URL: http://knowledgedefinednetworking.org/.

[13] Yan Chen et al. "An algebraic approach to practical and scalable overlay network monitoring". In: *Proceedings of the 2004 conference on Applications technologies architectures and protocols for computer communications SIGCOMM 04* 34.4 (2004), p. 55.

[14] Bo Han et al. “Network function virtualization: Challenges and opportunities for innovations”. In: *IEEE Communications Magazine* 53.2 (2015), pp. 90–97.

[15] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. “Improving the scalability of data center networks with traffic-aware virtual machine placement”. In: *Proceedings - IEEE INFOCOM* (2010), pp. 1–9.

[16] Tatsuaki Kimura et al. “Spatio-temporal factorization of log data for understanding network events”. In: *Proceedings - IEEE INFOCOM*. IEEE, 2014, pp. 610–618.

[17] Ian F Akyildiz et al. “5G roadmap: 10 key enabling technologies”. In: *Computer Networks* 106 (2016), pp. 17–48.

[18] Aref Meddeb. “Internet QoS: Pieces of the puzzle”. In: *IEEE Communications Magazine* 48.1 (2010), pp. 86–94.

[19] Joan Bruna et al. “Spectral Networks and Locally Connected Networks on Graphs”. In: *arXiv:1312.6203* (2013).

[20] Dan W. Patterson and Dan W. *Artificial neural networks : theory and applications*. Prentice Hall, 1996, p. 477.

[21] Ryszard Michalski, Jaime G Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science \& Business Media, 2013.

[22] Bruno Nunes Astuto et al. “A Survey of Software-Defined Networking : Past , Present , and Future of Programmable Networks”. In: *IEEE Communications Surveys and Tutorials* 16 (2014), pp. 1617–1634.

[23] N M Mosharaf Kabir Chowdhury and Raouf Boutaba. “A survey of network virtualization.pdf”. In: *Computer Networks* 54.5 (2010), pp. 862–876.

[24] Gunter Bolch et al. *Queueing Networks and Markov Chains*. Wiley-Interscience, 2006, pp. 1–878.

[25] Dean L Isaacson, Richard W Madsen, and John Wiley. “Markov Chains Theory and Applications”. In: *New York: Wiley* 4 (1976).

[26] L. Kleinrock. *Queuing systems, vol. 1: Theory*. Ed. by Wiley-interscience. 1975.

[27] James R. Jackson. “Networks of Waiting Lines”. In: *Operations Research* 5.4 (1957), pp. 518–521.

[28] J R Jackson. “Jobshop-Like Queueing Systems”. In: *MANAGEMENT SCIENCE* 10.1 (1963), pp. 131–142.

[29] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Vol. 2050. Springer, 2004, pp. xix –274.

[30] OpenSim Ltd. *OMNeT++ Discrete Event Simulator System*. URL: https://omnetpp.org/.

[31] Kihong Park and Walter Willinger. *Self-Similar Network Traffic and Performance Evaluation*. Vol. 53. New York, USA: John Wiley & Sons, Inc., 1989, p. 160.

[32] Will E. Leland, Murad S. Taqqu, and Daniel V. Wilson. “On the Self-Similar Nature of Ethernet Traffic (Extended Version)”. In: *IEEE/ACM Transactions on Networking* 2.1 (1994), pp. 1–15.

[33] Romain Fontugne et al. "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking". In: *ACM CoNEXT '10* (2010), p. 12.

[34] CAIDA. *Center for Applied Internet Data Anaysis Database*. URL: https://www.caida.org/data/overview/.

[35] ITA. *Traces available in the Internet Traffic Archive*. URL: http://ita.ee.lbl.gov/html/traces.html.

[36] Pere Barlet-Ros et al. "Predictive Resource Management of Multiple Monitoring Applications". In: *IEEE/ACM Transactions on Networking* 19.3 (2011), pp. 788–801.

[37] Peter G. Zhang. *Time series forecasting using a hybrid ARIMA and neural network model*. 2003. URL: http://linkinghub.elsevier.com/retrieve/pii/S0925231201007020.

[38] C Narendra Babu and B Eswara Reddy. "A moving-average filter based hybrid ARIMA–ANN model for forecasting time series data". In: *Applied Soft Computing Journal* 23 (2014), pp. 27–38.

[39] Chih Fong Tsai et al. "Intrusion detection by machine learning: A review". In: *Expert Systems with Applications* 36.10 (2009), pp. 11994–12000.

[40] T T T Nguyen and G Armitage. "A survey of techniques for internet traffic classification using machine learning". In: *Communications Surveys & Tutorials, IEEE* 10.4 (2008), pp. 56–76.

[41] Joe Wenjie Jiang et al. "Joint VM Placement and Routing for Data Center Traffic Engineering". In: *INFOCOM, Proceedings IEEE* (2012), pp. 2876–2880.

[42] J. E. Jr. Dennis and R. B. Schnabel. *Numerical Methods For Un- constrained Optimization and Nonlinear Equations*. Ed. by Prentice-Hall. 1983.

[43] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980* (2014).

[44] Albert Mestres et al. "Knowledge-Defined Networking". In: (2016), pp. 1–8.

[45] Dimitri P Bertsekas. *Network optimization: continuous and discrete models*. Athena Scientific Belmont, 1998.

[46] Ning Wang et al. "An overview of routing optimization for internet traffic engineering". In: *IEEE Communications Surveys & Tutorials* 10.1 (2008).

[47] Gyula Simon et al. "Simulation-based optimization of communication protocols for large-scale wireless sensor networks". In: *IEEE aerospace conference*. Vol. 3. 2003, pp. 31339–31346.

[48] Ian F Akyildiz et al. "A roadmap for traffic engineering in SDN-OpenFlow networks". In: *Computer Networks* 71 (2014), pp. 1–30.

[49] Giovanni Giambene. *Queuing theory and telecommunications*. Springer, 2005.

[50] Alberto Dainotti, Antonio Pescape, and Kimberly C Claffy. "Issues and future directions in traffic classification". In: *IEEE network* 26.1 (2012).

[51] Robin Sommer and Vern Paxson. "Outside the closed world: On using machine learning for network intrusion detection". In: *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE. 2010, pp. 305–316.

[52] He Yan et al. "G-rca: a generic root cause analysis platform for service quality management in large ip networks". In: *IEEE/ACM Transactions on Networking* 20.6 (2012), pp. 1734–1747.

[53] Shih-Chun Lin et al. "QoS-aware adaptive routing in multi-layer hierarchical software defined networks: a reinforcement learning approach". In: *Services Computing (SCC), 2016 IEEE International Conference on*. IEEE. 2016, pp. 25–33.

[54] George EP Box et al. *Forecasting and time series analysis*. John Wiley \& Sons, 2015, p. 304.

[55] Hongye Gao. "Identification and hypothesis testing on ARIMA (p, d, q) models". In: *Acta Mathematicae Applicatae Sinica* 5.1 (1989), pp. 15–20.

[56] C. Charalambous. "Conjugate gradient algorithm for efficient training of artificial neural networks". In: *IEE Proceedings G (Circuits, Devices and Systems)* 139.3 (2004), pp. 301–310.

[57] M.T. Hagan and M.B. Menhaj. "Training feedforward networks with the Marquardt algorithm". In: *IEEE Transactions on Neural Networks* 5.6 (1994), pp. 989–993.

[58] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural Networks* 12.1 (1999), pp. 145–151.

[59] Ben Pfaff et al. "The Design and Implementation of Open vSwitch". In: *NSDI'15 Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (2015), pp. 117–130.

[60] M Roesch. "Snort: Lightweight Intrusion Detection for Networks." In: *LISA '99: 13th Systems Administration Conference* (1999), pp. 229–238.