Farcast: Improving Forecasting Accuracy via SDN

Michael Bamiedakis-Pananos

Thesis submitted in partial fulfillment of the requirements for the

Master of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
University Campus, Voutes, Heraklion, GR-70013, Greece

Thesis Advisor: Assistant Prof. *Xenofontas Dimitropoulos*

This work was partially supported by **Institute of Computer Science, Foundation of Research and Technology Hellas**

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

Farcast: Improving Forecasting Accuracy via SDN

Thesis submitted by
Michael Bamiedakis-Pananos
in partial fulfillment of the requirements for the
Master of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Michael Bamiedakis-Pananos

Committee approvals: _____
Xenofontas Dimitropoulos
Assistant Professor, Thesis Supervisor

_____
Maria Papadopouli
Associate Professor, Committee Member

_____
Athanasios Mouchtaris
Associate Professor, Committee Member

Departmental approval: _____
Antonis Argyros
Professor, Director of Graduate Studies

Heraklion, June 2017

Abstract

Forecasting in computer networks is the process of anticipating future traffic demands based on present and past data and by analysis of trends. Network providers use forecasting in order to examine whether the traffic is routed efficiently, also they use it for bandwidth allocation and congestion control. In this work, we propose an approach, which we call Farcast, for improving the accuracy of network traffic load forecasting using Software Defined Networking (SDN) principles. In Farcast, the traffic load predictions for monitored links are sent to an SDN controller. The SDN controller has a global view of the network and monitors the network state in time frames called *inform intervals*. In these intervals, the SDN controller receives messages from network switches, about the current traffic flows traversing the network. Then, the SDN controller assesses whether the predictions will be accurate or not, based on the current traffic demands. If there are imminent inaccurate predictions, the controller ensures that the prediction error is reduced, by re-routing, when possible, the flows in the network accordingly.

Farcast was implemented in a simulated SDN environment. The experiments simulate traffic bursts that are difficult to predict with traditional forecasting approaches. We compare the performance of the AutoRegressive Integrated Moving Average (ARIMA) forecasting method with Farcast. The results show that Farcast reduces the Mean Absolute Percentage Error (MAPE) of the traffic load predictions, by up to one order of magnitude, when the goal is to satisfy the predictions for a single link in the network. Moreover, when adjusting the load of multiple monitored links concurrently, Farcast reduces the MAPE by up to 50%. Our approach can be deployed independently of the machine learning algorithm used for the predictions.

For example, such a approach would be useful in the 40-Gigabit-capable Passive Optical Network (XG-PON) standard. The PON is composed of Optical Netwok Units (ONUs) and an Optical Line Terminator (OLT). The ONUs predict the future traffic load and send bandwidth requests to the OLT, based on their predictions. The OLT performs the dynamic bandwidth allocation based on the ONUs' requests. In such schemes, the dependability on forecasting is critical, as wrong predictions can lead to underutilization of system resources or traffic congestion.

Περίληψη

Το Forecasting στα δίκτυα υπολογιστών είναι η διαδικασία πρόβλεψης μελλοντικών αιτημάτων κυκλοφορίας με βάση τα υπάρχοντα και τα παρελθόντα δεδομένα και την ανάλυση των τάσεων. Η πάροχοι δικτύου χρησιμοποιούν το Forecasting ώστε να εξετάσουν εάν η κίνηση διοχετεύεται αποτελεσματικά, χρησιμοποιούν τις προβλέψεις για την κατανομή εύρους ζώνης και τον έλεγχο συμφόρησης. Για τέτοιου είδους εφαρμογές, η ακρίβεια των προβλέψεων για την μελλοντικό φόρτο κίνησης, έχει μεγάλη σημασία για τους παρόχους δικτύου. Στη παρούσα διπλωματική, προτείνουμε μια προσέγγιση, την οποία ονομάζουμε Farcast, για τη βελτίωση της ακρίβειας της πρόβλεψης της κίνησης του δικτύου χρησιμοποιώντας τις αρχές των Δικτύων Καθοριζόμενων από Λογισμικό (SDN). Στο Farcast, οι προβλέψεις που έγιναν από τους μεταγωγείς δικτύου αποστέλλονται σε έναν ελεγκτή SDN. Ο ελεγκτής SDN έχει μια συνολική εικόνα του δικτύου και είναι σε θέση να παρακολουθεί τη κατάσταση του δικτύου σε ορισμένα χρονικά πλαίσια που ονομάζονται διαστήματα ενημέρωσης. Σε αυτά τα διαστήματα, ο ελεγκτής SDN λαμβάνει μηνύματα από τους μεταγωγείς δικτύου, σχετικά με τις τρέχουσες ροές κυκλοφορίας που διέρχονται από το δίκτυο. Έπειτα, ο ελεγκτής SDN αξιολογεί εάν οι προβλέψεις των μεταγωγέων πρόκειται να είναι ακριβείς ή όχι, βασιζόμενος στις τωρινές ανάγκες της κίνησης. Αν υπάρχουν ανακριβείς προβλέψεις που αναμένονται, εξασφαλίζει ότι το σφάλμα πρόβλεψης για κάθε μεταγωγέα μειώνεται, αναδρομολογώντας, όταν αυτό είναι δυνατόν, τις υπάρχουσες ροές στο δίκτυο ανάλογα.

Το Farcast υλοποιήθηκε σε ένα περιβάλλον προσομοίωσης SDN. Τα πειράματα προσομειώνουν απότομες μεταβολές της κίνησης, οι οποίες είναι δύσκολο να προβλεφθούν με τις παραδοσιακές μεθόδους πρόβλεψης. Συγκρίνουμε την απόδοση της μεθόδου πρόβλεψης Αυτοπαλίδρομου Ολοκληρωμένου Μοντέλου Κινητού Μέσου ARIMA με το Farcast. Τα αποτελέσματα δείχνουν ότι το Μέσο Απόλυτο Ποσοστιαίο Σφάλμα (MAPE) μπορεί να υποστεί μείωση μιας τάξης μεγέθους όταν ο στόχος είναι να ικανοποιηθούν οι προβλέψεις μιας σύνδεσης εντός του δικτύου. Επιπλέον, όταν παρακολουθείται ο φόρτος πολλαπλών συνδέσεων ταυτόχρονα, υπάρχει βελτίωση στο MAPE, έως 50%. Η μέθοδος μας μπορεί να υλοποιηθεί ανεξαρτήτως του αλγορίθμου μηχανικής μάθησης που χρησιμοποιείται για την εξαγωγή προβλέψεων.

Για παράδειγμα, μια τέτοια προσέγγιση θα ήταν χρήσιμη στο παθητικό οπτικό δίκτυο χωρητικότητας 10 Gigabit (XG-PON). Το PON αποτελείται από οπτικές δικτυακές μονάδες (ONUs) και μια γραμμή οπτικού τερματισμού (OLT). Τα ONUs προβλέπουν το μελλοντικό φόρτο της κίνησης και στέλνουν αιτήματα εύρους ζώνης στην OLT, τα οποία βασίζονται στις προβλέψεις τους. Η OLT κατανέμει δυναμικά το εύρος ζώνης βασιζόμενο στα αιτήματα των ONUs. Σε τέτοια σενάρια, η αξιοπιστία των προβλέψεων είναι μείζονος σημασίας, καθώς λάθος προβλέψεις μπορεί να οδηγήσουν σε υπολειτουργία των πόρων του συστήματος ή σε κυκλοφοριακή συμφόρηση.

Acknowledgements

Fist, I would like to thank my Professor, Xenofontas Dimitropoulos, for his valuable guidance and encouragement during the course of this thesis. I would like to thank him for giving me the opportunity to work with him and making me a member of the INSPIRE group.

Then, I would like to specially thank Christos Liaskos, to whom I am grateful, for all of his support, encouragement, contribution to this work, advices and excellent ideas.

I would like to thank all of the members of the group Manos, George, Vasileios, Pavlos, Lefteris, Dimitris, Alexandros, Petros and Gabriel for their constant encouragement and helpful discussions during the course of this thesis.

Last but not least, I would like to devote this work to my family and friends whose support and belief in me was unparalleled.

The project leading to this application has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 338402.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Forecasting is the process of making predictions for the future based on past and present data and most commonly by analysis of trends. Network traffic analysis and prediction resembles a proactive approach instead of a reactive approach, where a network is monitored for security breaches or other malicious or out-of-order behaviour [1]. Internet traffic forecasting can help to detect such misbehaviours of the network traffic, before they actually occur. Security attacks like Denial-of-Service, viruses, crossfire attacks [2] or spam could be predicted by comparing the real traffic with the predicted traffic of forecasting algorithms [3]. This results in an earlier detection of attacks which can benefit the quality of service (QoS).

Internet traffic forecasting is of great benefit in areas like bandwidth allocation, network security, network planning and predictive congestion control [4]. Long-period traffic forecasting is used for designing future capacity requirements by Internet-Service-Providers (ISPs) and permits for better management decisions [5]. With better traffic forecasting, reliable traffic engineering tools can be made, that adapt to future conditions of the network [6].

On the other hand, short-period prediction could improve the dynamic resource allocation and can be used to improve the QoS mechanisms [7]. These network schemes help in distributing the network bandwidth with regard to the predicted traffic [8]. Predictive bandwidth allocation can be used in data centers in order to improve the link utilization [9] and in ATM networks [10]. As an example in predictive bandwidth allocation, consider the 40 Gigabit Passive Optical Network (40G-PON or XG-PON), which is a networking standard for data links capable of delivering rates up to 40 Gbits/s over existing fibre. These networks consist of an optical line termination (OLT) and several optical network units (ONUs), as shown in Figure 1.1. The ONUs send traffic to the OLT, in each time slot called frame. The OLT provides transmission slots to the ONUs, by performing a Dynamic Bandwidth Allocation (DBA) algorithm after receiving requests from ONUs.

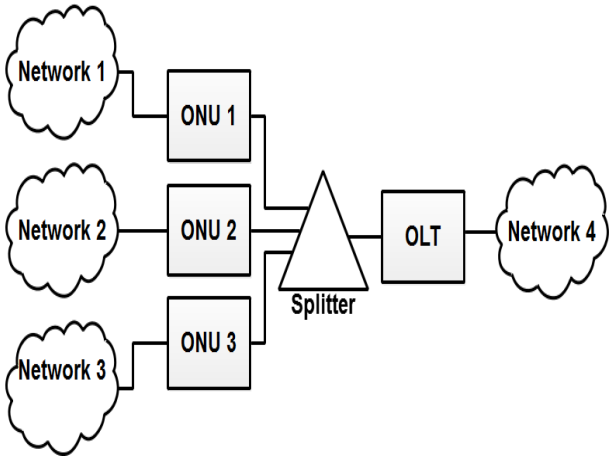


Figure 1.1: XG-PON network.

.

Because of the fibre's differential distance of up to 40 km between the OLT and the ONU's, there are high propagation delays which should be taken into consideration. There are works that develop fair algorithms for bandwidth allocation [11]. Interestingly, there is recent work in which the authors create a DBA scheme for XG-PON, that relies on forecasting the additional arrivals in ONUs during the polling process [12]. By forecasting the additional arrivals, the additional bandwidth requests are predicted, making improvements to the network performance in latency and jitter.

However, traffic forecasting can be inaccurate. The traffic spikes that may occur are difficult to detect even when the traffic is directly observed [13], let alone when someone has to predict them. The traffic spikes have an negative impact in bandwidth applications. For example in the XG-PON case, traffic spikes will result in poor bandwidth allocation, as the bandwidth requests are based on the ONUs predictions on their future traffic [14].

1.2 Thesis Contribution

This work proposes an approach, which we call Farcast, to improve the network traffic forecasting accuracy, independently of the traffic model or the method used for forecasting by the network's components. Our goal is to improve the Internet traffic forecasting accuracy, using Software Defined Networks (SDN). The SDN controller is the SDN's brain and has a global view of the whole network, therefore it has all the information it needs about the traffic that traverses the network and the network's topology. Thus, the controller is the most suitable candidate for traffic shaping and traffic engineering, which with a proper scheme, can result to more dependable Internet traffic forecasting.

Farcast is implemented in the OMNeT++ network simulator [15], using the OfOMNeT framework [16]. We simulate a computer network and make predictions on specific links

using the Auto Regressive Integrated Moving Average (ARIMA) method [17]. Then, we simulate traffic bursts by inserting into the network more traffic than the ARIMA anticipates or by extracting traffic from the network. In the results we compare the performance of the ARIMA method with the performance of Farcast. The results show that Farcast can improve the Mean Absolute Percentage Error (MAPE) by one magnitude of order when one link is monitored. Furthermore, when several links are monitored and there are no back-up links to extract or insert traffic, we show that Farcast can improve the MAPE by 50%.

As an example of an application that Farcast could be useful, consider the XG-PON standard. Farcast can redistribute the existing traffic in order to help the ONUs make more accurate predictions, by redirecting traffic from an ONU that made a bandwidth request that is higher than the bandwidth the ONU needs, to an ONU that made a bandwidth request that is lower than the bandwidth it needs. In this way, the network performance will be further improved.

1.3 Thesis Outline

The rest of this chapter presents the related work on forecasting. It contains work on improving forecasting accuracy using different machine-learning methods. The rest of this thesis is written based on the following outline.

Chapter 2 presents the Software Defined Networks and the OpenFlow protocol. Additionally, it shows the basic principles of time-series forecasting (TSF) using the ARIMA method. Furthermore, it presents the OMNeT++ simulator and the OfOMNeT framework which were used at the implementation.

Chapter 3 details the methodology of this work. It explains the intervals used for Farcast monitoring and the additions that were implemented at the switches and the controller respectively, in order to support Farcast. Moreover, it explains how the controller makes its re-routing decisions. Finally, in this Chapter the limitations of Farcast are examined in detail.

In Chapter 4 the setup used for evaluating Farcast's performance is shown. A discussion of the results is included.

Finally, the conclusion and future work directions are presented in Chapter 5.

1.4 Related Work

Internet traffic forecasting has been studied thoroughly, because of the topic's interest and importance. In [18, 19] the authors show that the ad hoc network traffic and the Ethernet

traffic, respectively, is self-similar. Particularly, in [18] they validate that the ad hoc network traffic is forcastable, since self-similar time-series can be forecasted. Additionally, in [20] the authors assess the predictability of network traffic and they conclude that network traffic forecasting shows good potential. This potential is also shown in [21], where the author focus on the predictability of the TCP/IP end-to-end throughput and latency.

The authors in [3] and [22] compare the ARIMA and Holt-Winters methods with a novel neural network ensemble approach on the same datasets, to assess which method works best for different time frames. They concluded that the Holt-Winters is the best option with the daily forecasts, while the neural ensemble achieved best results for 5 minute and hourly datasets. Additionally, they show that the neural network ensemble is competitive compared with the Holt-Winters and the ARIMA.

In [4] the authors experiment with forecasting approaches for Internet traffic and propose different models and architectures like combining FARIMA and Artificial Neural Networks (ANN). They select their model based on the White's Neural Network test for non-linearity and compare their hybrid forecasting models with well-known methods, such as Holt-Winters, ARIMA, and FARIMA. They find that forecasting approaches which take non-linearity into account lead to better overall forecasts for Internet traffic.

The authors in [8], present a Neural Network method for traffic forcasting for all links of a backbone network, using both univariate and multivariate strategies. The former uses only past values of the forecasted link, while the latter combines past values of the forecasted link and the past values of the neighbouring links that are expected to affect the link which traffic is forecasted.

The authors in [23] introduce a multi-resolution finite-impulse-response neural-network-based learning transform. This learning algorithm employs the analysis of a signal into wavelet coefficients and scaling coefficients. When this algorithm is applied to network traffic prediction, the results show that the accuracy of the neural network is improved.

In [5], the authors propose a new method for collecting historical data from Internet data flows. Furthermore, they present the use of new long-term forecasting models based on multiple settings of Time-Lagged-Feedforward Networks, which is a static neural network, whose model includes a delay window that works as a short-term memory. Results show that this approach is a good option for planning network links that transport Internet traffic.

Additionally, in [24], the authors propose another model for long-term traffic prediction by dealing with the internal relationship of long time scale network traffic, this paper combines the regular trend and the smooth or seasonal trend of hours and days, then fits the dual-related model to predict long time scale traffic. The result indicates that the proposed model effectively identified the correlations of data between days and hours, and is successful in forecasting approaches.

In [25], the authors develop a methodology to forecast the fluctuations of Internet traffic in an international IP transit network. If needed, the origin-destination demands are estimated a posteriori through traffic matrix inference techniques. Their methodology relies on Principal Component Analysis and Time Series modeling. They show that five components represent most of the traffic total variance and that these components are quite stable over time. This stability allows them to develop a method that produces forecasts automatically without any model to fit.

All previous works mentioned above focus on making the Internet traffic predictions more accurate by improving the methods or generating new models/techniques that are used for traffic prediction. Additionally, they compare how different models make predictions in different situations like short-term, mid-term or long-term predictions to assess which model is best for each situation. The difference with our work is that it is focused on making the traffic predictions accurate independently of the prediction algorithm used by using the Software Defined Network's global view of the network to reroute flows.

In [7] the authors advocate the use of time series analysis for Web traffic modeling and forecasting. They show that the number of Web requests handled by a server or the amount of data retrieved per hour by a server can be accurately modeled with seasonal ARIMA models. Then, they use these models to make medium-term predictions of client requests characteristics. The predictions are taken into consideration for dimensioning decisions.

The authors in [26], present a novel network attack diagnostic methodology, based on the characterizations of the dynamic statistical properties of normal network traffic. They develop an anomaly-tolerant non-stationary traffic prediction technique and then introduce dynamic thresholds where they define adaptive anomaly violation conditions as a combined function of magnitude and duration of the traffic deviations. They show that the approach is efficient and effective under the presence of different attacks such as mail-bombing attacks and UDP flooding attacks. While this work is mostly statistical, it shows how powerful the network traffic prediction is for the detection of network attacks.

Adaptive traffic engineering depends on the ability to obtain accurate snapshots of the network's state in little time, as well as to react rapidly to state changes. In such traffic engineering methods, a control server periodically measures the traffic load in the network and dynamically changes the routes so as to minimize the network congestion [27, 28, 29, 30, 31]. However, traffic engineering using the measured traffic only mitigates the observed congestion and never avoids the future congestion. The difference with our work is that we replace the computer server with a SDN controller. Additionally, instead of measuring the traffic load and avoiding the congestion, we take into account the predictions about future traffic and ensure their validity, if they do not lead to congestion.

Predictive traffic engineering has the same goals with the adaptive traffic engineering, however the server monitors the traffic predictions instead of the actual traffic in the net-

work. In [6], the authors propose a prediction procedure that consider the short-term and longer-term future traffic demands, in order to adapt their traffic engineering decisions. They focus on the results of traffic engineering instead of the accuracy of the predictions, while we focus on ensuring the prediction's validity.

Chapter 2

Background

This chapter focuses on the background of this thesis. It contains a description of the Software Defined Networks architecture, which is used in Farcast for monitoring and re-routing flows to help the switches make more accurate predictions. In addition, it defines the time-series forecasting and describes the ARIMA methodology for predicting the next value in a time-series. Furthermore, it gives a description of the OMNeT++ simulator and the OfOMNeT framework, which is used for SDN simulations in OMNeT++.

2.1 Software Defined Networks

Traditional enterprise networks are characterized by high design complexity, which generally leads to a significant amount of administrative manual intervention. The more complex a network is, the more likely it is to be prone to failures, making it more difficult to upgrade and manage [32]. To configure the network's policies, the network operators need to set the configuration of each network device, separately, using low-level commands [33]. These commands are often vendor specific, making it even harder to set the network behaviour across different vendor hardware.

In addition, the control plane than manages the forwarding decisions and the data plane that forwards traffic according to the control plane's decisions are implemented together in the networking devices. This reduces flexibility making it harder for the researchers to make innovations at the networking infrastructure. Because of the network inflexibility, the Internet architecture is hard to evolve [34].

Software Defined Networking (SDN) is a network approach that allows the network operator to perform the administrative functions of controlling, changing and managing the network behaviour dynamically, by using open interfaces and abstractions of low-level functionality. SDN changes the network limitations by separating the network's control plane from the data plane. This makes the network switches inexpensive, since they just become simple forwarding devices, while all the control logic is placed in a centralized controller [35, 36].

Figure 2.1 shown a simplified view of an SDN architecture. The decoupling of the control plane and the data plane is shown, as the forwarding devices, e.g. the OpenFlow switches, are separated from the controller platform. The Southbound APIs are used for the SDN controller and forwarding devices communications, e.g. the OpenFlow protocol. The southbound APIs define the way that the SDN controller interacts with the underlying infrastructure. Furthermore the northbound APIs are used for the SDN controller to interact with the services and applications running over the network. This gives the SDN network programmability. In simple terms, the northbound interface is an interface that allows the controller to communicate with higher level components, while the southbound interface allows the controller to communicate with lower-level components.

In SDN the forwarding decisions are flow-based. A flow is defined by a set of packet

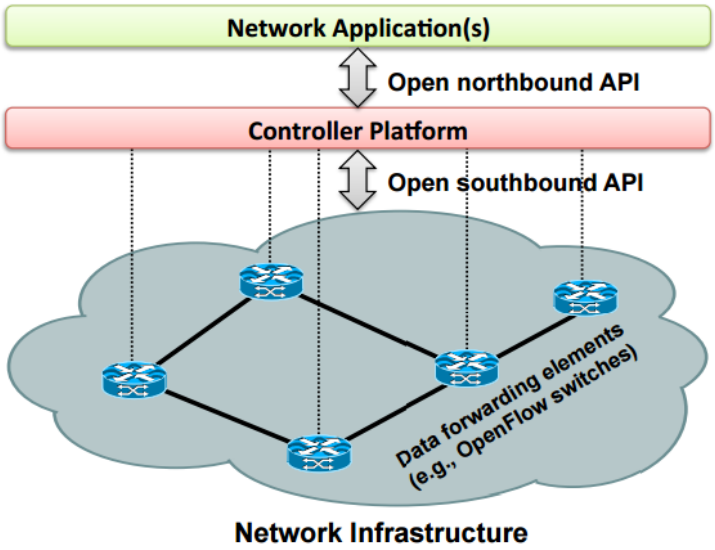


Figure 2.1: Simplified view of an SDN architecture. [33].

field values acting as a match criterion and a set of actions on the flow. The matching criteria group together a set of packets from a source to a destination. On this particular group, a set of actions, meaning instructions is implemented. Using this approach, the group of packets will receive similar service from the network regardless of the device that handles them, since the controller sets the behaviour of all the devices in the network. This flexibility allows for innovation as new protocols can be implemented easily, as well as, adaptability in situations were different control logic needs to be implemented by the network operators. Another plus of the SDN architecture is that the controller has a global view of the network, making it easier to make topology aware forwarding decisions [37, 38].

All the control logic is moved to an external entity, the SDN controller or Network Op-

erating System (NOS) [39]. The NOS allows the programming of the forwarding devices of the network and can be seen as the operating system of the network. On top of the NOS, several software applications are run. These applications interact with the network hardware through the logical abstractions and resources provided by NOS.

The limitations of SDN is that with the increasing number of forwarding devices in the network, the number of messages exchanged between the devices and the controller, increases. Additionally, the more distant a switch is, the more delay is added to the controller-switch communication. Moreover, the controller is a single point of failure in the network, if the controller is compromised by an external malicious entity, the whole network's security is also compromised. Finally, the controller's processing power bounds the performance of the network. These issues can be solved by having a cluster of controllers manage the network instead of just a single controller. Of course, adding more controllers raises the cost of deploying such a network.

The OpenFlow standard is managed by the Open Networking Foundation, a user-led organization dedicated to the promotion of Software Defined Networking. The Openflow is a communications standard between the control and forwarding layer of the SDN architecture. OpenFlow allows the direct access and manipulation of the forwarding plane of network devices and allows switches from different vendors to be managed using a single open protocol. The OpenFlow is layered on top of the Transmission Control Protocol (TCP) and controllers listen on TCP port 6653 for switches that want to set up a connection.

The SDN architecture provides the network operators with flexibility, adaptability and innovation. SDN is expected to be adopted by a majority of the enterprises and its market is expected to grow to $12.5 billion by 2020 [40], which is another indicator of its overall contribution to networks' performance.

2.2 ARIMA

A time series is an successive order sequence of values of a variable at equally spaced time intervals [17]. A time series tracks the movement of chosen data points such as the the amount of traffic forwarded by a switch over a specified period of time with data points recorded at regular intervals.

Time series analysis can by used to see how a variable changes over time. It can also be used to examine how the changes are associated with the chosen data compare to shifts in other variables over the same time period.

Alternative, one can search for patters of seasonality in a time-series to find out if there is a pattern in situations exhibiting dependency between the data points and the chosen variable. A time series model assumes that past patterns will occur in the future. Another

relevant concept is the horizon or lead time, which is defined as the time in advance that a forecast is issued.

Seasonality, [41], in a time series is a regular pattern of changes that repeats over S time periods, where S defines the number of time periods until the pattern repeats again. However non seasonal behaviour will still matter as with seasonal data it is likely that short run non-seasonal components will still contribute to the model.

Time series forecasting (TSF) uses information regarding historical values and associated patterns to predict future activity. To make a time series forecast is to infer the probability distribution of a future observation from the population, given a sample z of past values. The goal of TSF is to model a black-box predicting the series' behaviour based on historical data and not how it works. The performance of a forecasting model is evaluated by an accuracy measure such as the sum squared error (SSE) and the mean absolute percentage error (MAPE):

$$e_t = y_t - \hat{y}_{t-h} \tag{2.1}$$

$$SSE = \sum_{i=P+1}^{P+N} e_i^2 \tag{2.2}$$

$$MAPE = \frac{1}{N} \sum_{P+1}^{P+N} \frac{|e_i|}{y_i} \times 100\% \tag{2.3}$$

where:

- e_t denotes the forecasting error at time t.
- y_t the desired value.
- $\hat{y}_{t,p}$ the predicted value for period t and computed period p.
- P is the present time.
- N the number of forecasts.

The $MAPE$ is a common metric in forecasting applications [17, 5, 42, 43, 22, 4] and it measures the proportionality between the forecasting error and the actual value. Reported disadvantages of MAPE are associated with instabilities, when the original time series carries small values, and with asymmetrical penalties applied to positive errors compared to the negative ones [4]. In [44], the authors write that, although the MAPE has become an industry standard, it is advisable to also consider other summary measures.

Another useful metric in forecasting applications is the SMAPE [45]. This metric is defined by equation :

$$SMAPE = \frac{1}{n} \sum_{t=1}^{n} \frac{|F_t - A_t|}{(|A_t| + |F_t|)/2} \tag{2.4}$$

However, in this thesis, only the $MAPE$ will be taken into consideration, as both the $MAPE$ and $SMAPE$ provide similar results in terms of comparing a network's prediction accuracy with and without Farcast.

Within the Forecasting community the following forecasting types can be defined, depending on the time scale [46, 22]:

- Long-term forecasting, meaning one to several months of years which is useful for management and financial decisions.
- Middle-term, typically from one to several days used to plan resources.
- Short-term, from one to several hours, crucial for optimal control or detection of abnormal situations.
- Real-time, which concerns samples not exceeding a few minutes and requires an on-line forecasting system.

This thesis will take into consideration only the last case, because of the time required to simulate the network and the flows traversing through the network. The Autoregressive Integrated Moving-Average (ARIMA) is an important forecasting approach that goes through model identification, parameter estimation and model validation [17, 22]. A model is a description of a system using mathematical concepts and language. The ARIMA model is based on a linear combination of past values (AR components) and errors (MA components).

The AR part of ARIMA indicates that the evolving variable of interest is regressed on its own lagged (prior) values. Linear regressions is an approach for modelling the relationship between a scalar dependent variable and one or more explanatory variables. The MA part indicates that the regression error is actually a linear combination of error terms whose values occurred contemporaneously and at various times in the past. The I indicates that that the data values have been replaced with the difference between their values and the previous values (this differencing may take place multiple times). The purpose of these features is to make the model fit the data as well as possible.

The non seasonal model is denoted by the form ARIMA (p,d,q), where parameters p,d and q are non negative numbers. p is the number of time lags of the autoregressive model, d is the degree of differencing (the number of times the data have had past values subtracted) and q is the order of the moving-average model. The ARIMA model can be estimated using the Box-Jenkins approach [17]. We denote y the d^{th} difference of Y:

$$y = \begin{cases} Y_t, & \text{if } d = 0 \\ Y_t - Y_{t-1}, & \text{if } d = 1 \\ (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2}), & \text{if } d = 2 \end{cases} \tag{2.5}$$

The general forecasting equation is:

$$\hat{y_t} = \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} - \theta_1 e_{t-p} - \ldots - \theta_q e_{t-q} \tag{2.6}$$

Where:

- $\hat{y_t}$ is the predicted value.
- θ's and ϕ's are estimated.
- e is white noise.

2.3 OMNeT++

OMNeT++ [47, 48, 15] is an object-oriented modular discrete event network simulator framework. OMNeT++ stands for Objective Modular Network Testbed in C++ and its architecture is generic and supports multiple problem domains such as:

- protocol modeling
- modeling of wired and wireless communication networks
- multiprocessors and other distributed or parallel systems
- modeling hardware architectures
- any system where the discrete event approach is suitable.

In OMNeT++ each component of a simulated system is implemented in a module. The modules are reusable (if well-written) and can be combined in various ways to form different models. These modules are connected to each other via gates. The can communicate with each other with the use of messages, meaning the topology is dependent of the way in which the modules are connected, thus the simulation topology is parametrizable. The modules pass the messages through the gates and connections, and they can carry arbitrary data structures. The module's behaviour can be parametrized, thus a router module can be parametrized to simulate different vendor routers.

An OMNeT++ model is consisted of:

- NED language topology descriptions, that describe the network topology using modules, gates, connections and parameters.
- Message definitions in which different message types are defined.
- Simple module sources, which are C++ files.

The simulation system provides the following components:

- Simulation kernel: contains the code that manages the simulation. It is written in C++ and compiled into a shared or static library.
- User interfaces: OMNET++ user interfaces are used in simulation execution to provide debugging or batch execution of simulation.

A discrete event system is a system where state changes (events) happen at discrete instances in time, and events take zero time to happen. Between two consecutive events nothing happens, meaning there is no change that takes place in the system between two consecutive events. This is in contrast to continuous systems where state changes are continuous. Systems that can be viewed as discrete event systems can be modeled using discrete event simulation (DES). For example, the computer networks are viewed as discrete event systems. Take into consideration some events, such as:

- The start of a packet transmission.
- The end of the packet transmission.
- The expiry of a retransmission timeout.

This means that between two events, there is not other interesting event that changes the system's state. If we are interested in a single packet transmission, we would only care about two events, the start of the transmission and the end of the transmission. The time when events occur is called event timestamp, while in OMNeT++ the term arrival time is used. Time within the models is termed simulation time, opposed to real time or CPU time which refer to how long the simulation program has been running and how much CPU time it has consumed, respectively.

Discrete event simulation maintains a set of future events. The initialization step produces a set of events that ensure that the simulation starts, then inserts newly spawned events as the simulation progresses. The events are always processed in timestamp order to maintain causality, meaning that future events cannot interfere on earlier events. An event may trigger other events, for example a packet drop event may cause a packet resend event.

The simulation stops when there are no more events lefts to process in the simulation or when the simulation time expires because it reached a certain limit, before the simulation exits the user has the option to record statistics to output files.

2.3.1 Modeling Concepts

The active modules in OMNeT++ are called simple modules, they are written in C++, using the OMNeT simulation class library. The simple modules can be grouped together to form compound modules and, again, the compound modules can be grouped to form other compound modules, as shown in Figure 2.2. A gate is the input and the output interface of a module, there are output and input gates and they can be linked with a connection.

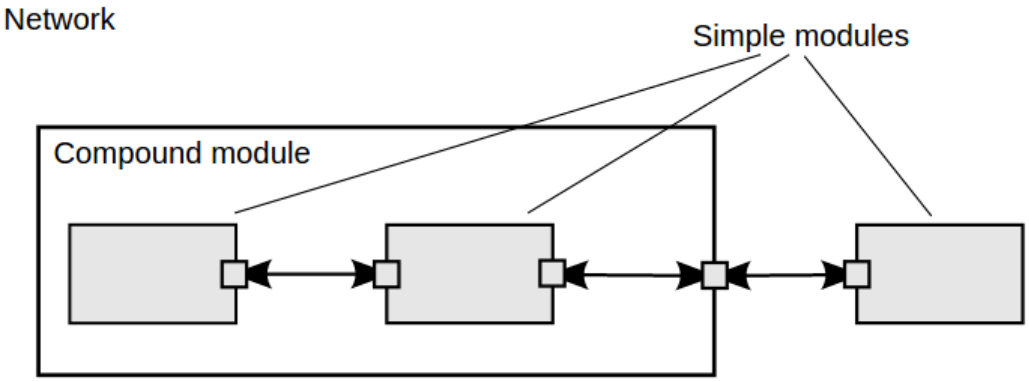


Figure 2.2: Simple and compound modules, [15].

An output gate sends a message and an input gate receives it. The connections can be created between the submodules of a module, between the module and the submodules and between the modules. The messages typically travel through a chain of connections starting and arriving in simple modules. The connections can be parametrized in terms of propagation delay, data rate and bit error rate. Also, the connection types can be defined and then reused as many times as necessary in the same or different simulation models. Furthermore, modules can have parameters which are mainly used to pass configuration data to simple modules and help in defining the network topology.

Parameters are variables that belong to a module. Parameters can be used in building the topology and to supply input to the C++ code that implements simple modules and channels. Parameters can be of type double, int, bool, string and xml. Parameters can get their values from NED files or from the configuration file (omnetpp.ini).

2.3.2 NED Overview

In OMNeT++, the user describes the simulation model in the Network Description (NED) language. The user declares simple modules and connects them in NED, he can also label some compound models as networks. Additionally, channels are a another component type that can, also, be used in compound modules. NED provides features that let it scale to large projects:

- Hierarchical: A single entity that is too complex can be broken down to smaller modules and used as a compound module.
- Component-Based: All modules, both simple and compound are reusable. This makes the components reusable without making copies of the same code. Additionally, it allows component libraries such as the INET framework to exist.
- Interfaces: Module and channel interfaces can be used as a placeholder where normally a module or channel type would be used.

- Inheritance: Modules and channels can be subclassed. The derived modules and classes may include new parameters and gates in addition to the inherited ones. It is also possible to set values to the existing parameters or set the gate size of a gate vector.
- Packages: The NED language features a Java-like package structure to reduce the risk of name clashes between different models.
- Inner types: Channel types and module types used locally by a compound module, can be defined within the module to reduce namespace pollution.
- Metadata annotations: Metadata can be used to carry extra information for various tools, the runtime environment or even for other modules in the module.

2.3.3 Events

OMNET++ uses messages to represent events. Messages are represented by instances of the $cMessage$ class and its subclasses. When a message is sent from one module to another, the place where the event will occur is the destination module of the message and the event time is the arrival time of the message. When a module wants to set a timeout expired event, it will send a message to itself. Events are consumed in arrival time order and more specifically:

- A first-come-first-served policy is follows on the execution of events, meaning that the earlier a message arrives the earlier it is executed. If two messages' arrival time is equal,
- events with the higher scheduling priority is executed, first. If priorities are the same,
- the message that was scheduled/sent earlier is executed first.

Scheduling priority is a user assigned integer attribute of messages.

2.4 Components, Simple Modules, Channels

OMNeT++ simulation models are composed of modules, that are simple and compound, and connections which may have associated channel objects. The channels and the models are programmable in C++ by the user. Modules and channels are represented with the $cModule$ and $cChannel$ classes, respectively. $cModule$ and $cChannel$ are both derived from the $cComponent$ class.

The user can define simple module types by subclassing $cSimpleModule$, while compound modules are established with the $cModule$, even though the user may override it. The $cChannel$'s subclasses include the three build-in channel-types: $cIdealChannel$,

$cDelayChannel$ and $cDatarateChannel$. By subclassing $cChannel$ or any other channel class, the user can create new channel types. The inheritance relationships between the modules are shown in Figure 2.3. Simulation signals are emitted by components and

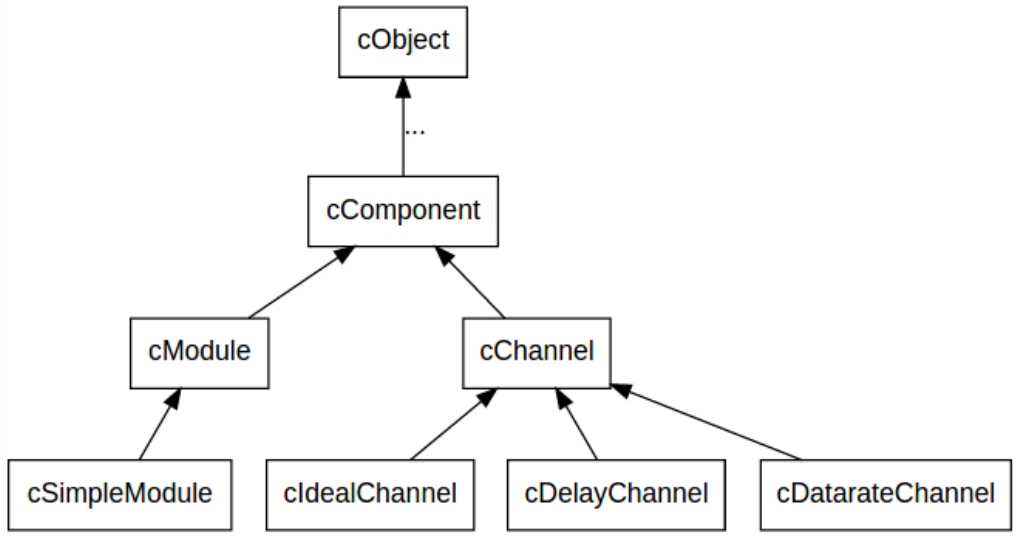


Figure 2.3: Inheritance of component, module and channel classes [15].

and can be used for:

- extracting statistical properties of the model without specifications of whether and how to record them
- receiving notifications about changes in the simulation model changes and acting upon them
- introducing a publish-subscribe communication between the modules
- emitting information for purposes like animation effects.

Signals propagate in the module hierarchy up to the root. At any level, the module can register listeners that will be notified when a signal value is emitted.

Signals are identified by signal names, however for efficiency dynamically assigned identifiers (IDs) are used. The signals are global, so all modules and channels resolving to a particular signal will get back the same numeric signal ID.

2.5 The INET Framework

The INET Framework [49, 16] is an open-source library for the OMNeT++ simulation environment. It provides protocols, agents and other models for researchers and students working with communication networks. INET is useful for designing and validating new protocols, or exploring new scenarios. Several other simulations frameworks take INET as a base and extend it into specific directions, such as vehicular networks, overlay/peer-to-peer networks, or long-term-evolution (LTE).

The INET Framework contains implementations of the protocols IPv4, IPv6, TCP, UDP, SCTP, UDP and several application models for the OMNeT++ simulator. The hosts and routers in the INET Framework are compound modules, composed of many ingredients. The ingredients of our interest are:

- Interface Table ($InterfaceTable$): This module contains the table of the network interfaces in the host. Interfaces are registered dynamically during the initialization phase by modules that represent network interface cards.
- Routing Table ($IPv4RoutingTable$): This module represents the IPv4 routing table. It contains member functions for adding, deleting, enumerating and looking up routes and finding the best matching route for a given destination IPv4 address.
- Network Interfaces: Network interfaces are compound modules composed of a MAC module and a queue. Some examples are the $EthernetInterface$ and the $WLAN$ interface.
- Network Layer: The INET Framework provides modules that represent protocols of the network layers that are usually grouped into a compound layer. IPv4 Network Layer for IPv4, and IPv6 Network layer for IPv6. The IPv4 Network layer contains the modules $IPv4$, ARP, $ICMP$ and $ErrorHandling$. The $IPv4$ module performs the IP encapsulation/decapsulation and routing of datagrams, using the IPv4 routing table C++ interface function call. The ARP module is put into the path of packets leaving the network layer towards the Network Interfaces and performs address resolution for interfaces like the Ethernet that need it. Finally, $ICMP$ handles the sending and receiving functions for the ICMP packets.
- Transport Layer protocols: Transport Layer is represented by modules that are connected to the network layer. The INET Framework supports TCP, UDP, SCTP modules.
- Applications: They typically connect to TCP or UDP and model the user behaviour and the application program's behaviour e.g. a browser and an application level protocol e.g. HTTP. The module $StandardHost$ supports any number of TCP, UDP and SCTP applications, their types being parametric.

The UDP protocol is a very simple datagram transport protocol which, essentially, connects the network layer to the applications. It performs packet multiplexing and demultiplexing to ports and some basic error detection. The frame format is shown in Figure 2.4.

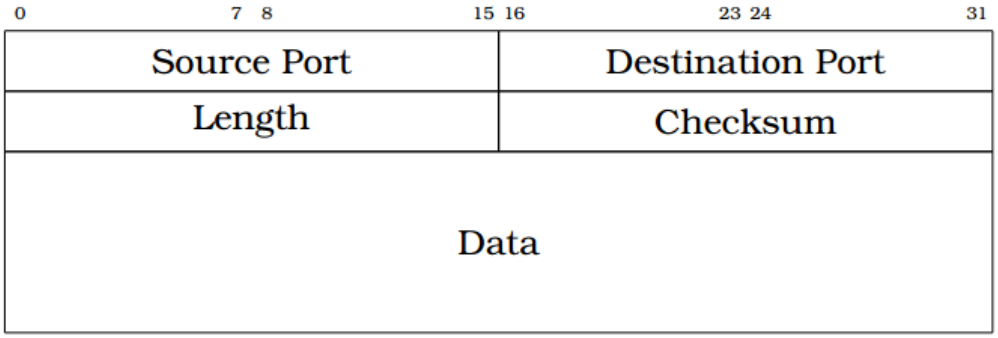


Figure 2.4: UDP datagram.

The ports represent the communication end points, allocated by the applications that need to send or receive the datagrams. The "Data" field is the encapsulated payload of the applications, the "Length" field is the length of the Data and the "Checksum" field is used to ensure "Data" integrity.

The INET framework contains a UDP module that performs the encapsulation/decapsulation of the application packets, a $UDPSocket$ class provides the application the socket interface and some sample applications. There are three important applications for this thesis the $UDPSink$, the $UPDBasicAPP$ and the $UDPBasicBurst$.

The $UDPsink$ binds a UDP socket to a given local port and prints the source, destination and length of each received packet.

The $UDPBasicApp$ sends UDP packets to the IP address given in the $destAddresses$ parameter. The application sends a message to one of the targets in each $sendInterval$ interval. Before a packet is sent, it is emitted in the $sendPK$ signal. The application simply prints the received UDP datagrams. The $rcvdPk$ signal can be used to detect the received packets. The number of sent and received messages are saved as scalars at the end of the simulation. The $UDPBasicApp$ has also the $starttime$ and $endtime$ parameters that specify the start and the end of the UPD flow's life.

The $UDPBasicBurst$ module sends UPD packets to the given IP addresses in bursts, or acts as a packet sink. It is compatible with both IPv4 and IPv6. It contains the following parameters:

- Addressing: The $destaddresses$ parameters can contain zero, one or more destination addresses, separated by spaces. If no IP addresses are given, the module will act as a packet sink. If there are more than one addresses, one of them is randomly chosen, either for the whole simulation run, or for each burst, or for each packet, depending on the value of the $chooseaddressmode$ parameter. The destination ad-

dress RNG parameter controls which random number generator (RNG) is used for randomized address selection. The self addresses are ignored. The peer can be a $UDPSink$ or another $UPDBasic$ Burst.

- Bursts: The first burst starts at $startTime$. Bursts start by sending a packet, immediately. Following the first packet more packets are sent at $sendInterval$ intervals. The $sendInterval$ parameter can be a random value. A constant interval with jitter can be specified. The length of the burst is set using the $burstDuration$ parameter (if a send interval value is greater than the burst's duration the burst will consist of only one packet). The time between two consequent bursts is the $sleepDuration$ parameter, which can be zero. Finally, the applications stop time is specified by the $stoptime$ parameter.
- Packets: Packet length is controlled by the $messageLength$ parameter. The module adds two parameters to packets before sending. These are the $sourceID$ and the $msgID$ which is incremented by 1 after any packet is sent.
- Sink Operation: When the $destAddresses$ parameter is empty, the module only receives packets and makes statistics.

2.6 OfOMNeT

In OfOMNeT [50], the OpenFlow components are integrated in the network simulation environment OMNeT++. Using the INET Framework, an OpenFlow switch and a basic controller are developed.

The OfOMNeT is the simulation model of the OpenFlow system in the INET framework for OMNeT++. In OfOMNeT the OpenFlow switch and the OpenFlow controller nodes are implemented according to the OpenFlow version 1.2 specifications [51]. Farcast implemented additional functionality in the OpenFlow controller, the OpenFlow switch modules as well as the OpenFlow controller behaviour application. The changes made to OfOMNeT for Farcast are discussed in chapter 4.1.

2.6.1 OpenFlow Messages

As shown in Figure 2.5, all the implemented messages are subclasses of the OFP_Header message, which includes the OpenFlow message header definition and the C++ class, so the modelisation of the messages is as close to the real messages as possible. The $OFP\_Feutures\_Request$ and the $OFP\_Feature\_Reply$ messages are sent during the initialization of the OpenFlow channel between the controller and a switch. The $OFP_{Packet_{In}}$ message is used to inform the controller about an unmatched incoming packet or to send a packet to the controller if this is the corresponding action of a match. The message sent to the controller can either contain the whole encapsulated packet or

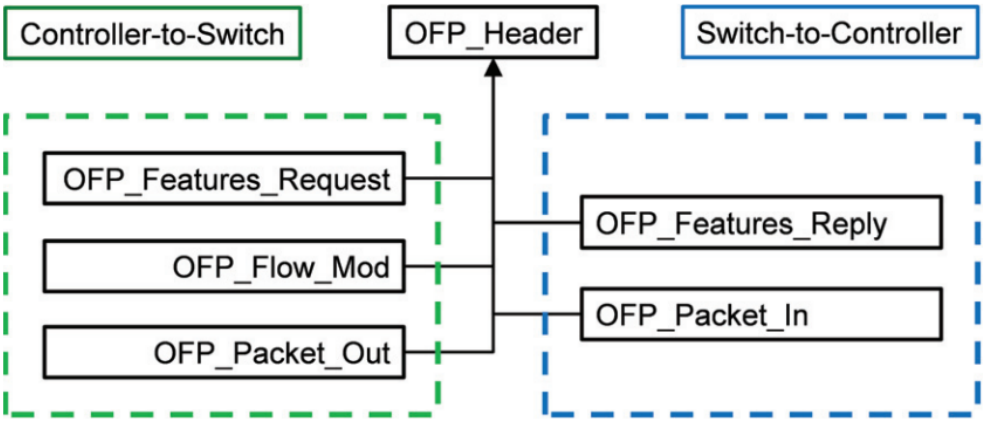


Figure 2.5: Implemented OpenFlow messages [50].

simply the buffer ID of the buffered packet. The OFP_Packet_Out message is sent by the controller to the switch. Using this message the controller instructs the switch to send a packet out of a specified switch port. Finally, the controller can modify the switch's flow table using the OFP_Flow_Mod message, which include the packet match field and the corresponding actions.

2.6.2 Openflow Switch

The separation of the data plane and the control plane implemented in OfOMNeT is shown in Figure 2.6. The data plane consists of an $EtherMAC$ module and an $OF_Processing$ module. The $EtherMAC$ module is connected to the outside and receives the incoming messages on the data plane. The received data frames are passed to the $OF\_Processing$ modules without modifications which implement the OpenFlow switch functionality on the data plane. This functionality comprises the required flow table lookups for all incoming packets, for possible matches or informing the OFA_Switch module on the control plane about packet-in events of unmatched packets. The $OF\_Processing$ module considers different complexities to the applied operations by applying a service time parameter which delays the simulation for a configured amount of time.

For inter-module communication between the data plane and the control plane in the switch, the OMNeT++ signal is used. The $OFA\_Switch\_Processing$ module emits a no-match-found signal and the OFA_Switch module subscribes to this signal. The OFA_Switch module implements the OpenFlow switch functionality on the control plane and is responsible for the communication with the controller. On module startup, the OFA_Switch module establishes a TCP connection to the controller and negotiates the supported OpenFlow version and capabilities. When a no-match-found event occurs the OFA_Switch module takes the unmatched packet and sends an OFP_Packet_In message for the controller. The message can contain either the full payload of the re-

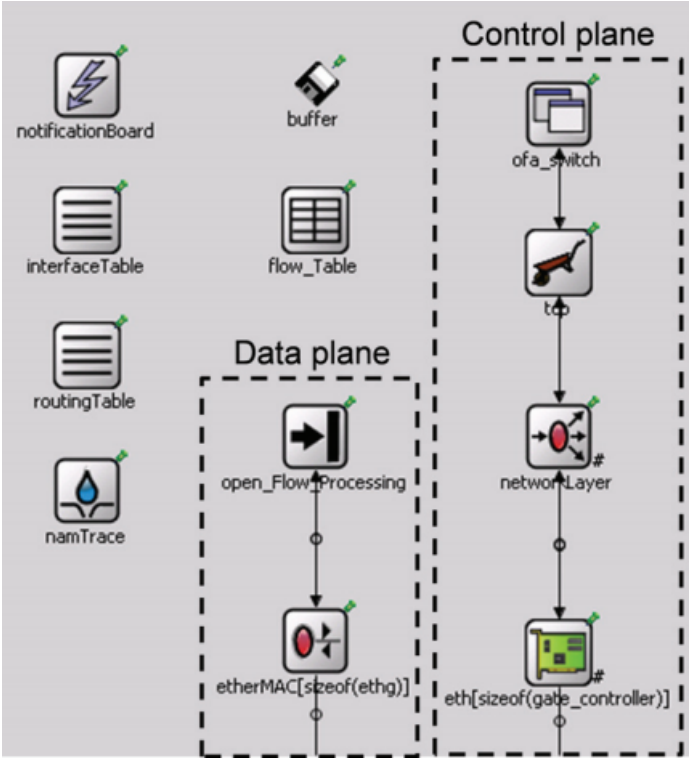


Figure 2.6: OfOMNeT Switch [50].

ceived packet or the buffer ID of the packet. If only the buffer ID is sent, the $Buffer$ module stores the packet until the reply from the controller arrives, in this case the reply will be an OFP_Packet_Out reply. The option of whether the packet is buffered or not is parametrizable and configured via a parameter in the switch.

The control plane of the OpenFlow switch in OfOMNeT has additional modules that are part of the TCP/IP stack and are required as the OFA_Switch module is modeled as TCP application because the OpenFlow channel uses a TCP connection (the channel between the controller and the switch). Through the OpenFlow channel the OFP_Packet_Out and OFP_Flow_Mod messages are transmitted from the controller. for each OFP_Flow_Mod message received, the OFA_Switch module extracts the embedded match and action and adds an entry to the $FlowTable$ module.

Finally, for the received OFP_Packet_Out messages, the OFA_Switch module needs to emit an event to the $OF\_Processing$ module, which will apply the corresponding output action. Specifically, this action can either be a flood-packet or a send-packet action. Therefore, the $OF\_Processing$ module is also subscribed to the associated actions signal. Again, the event may either contain the complete packet, or just the Buffer ID with which the packet can be retrieved from the $Buffer$ module.

2.6.3 Openflow Controller

Figure 2.7 shows the OpenFlow controller architecture in OfOMNeT. The control plane part consists of TCP/IP applications applications, similar to those of the OpenFlow switch, responsible for the controller-switch communications.

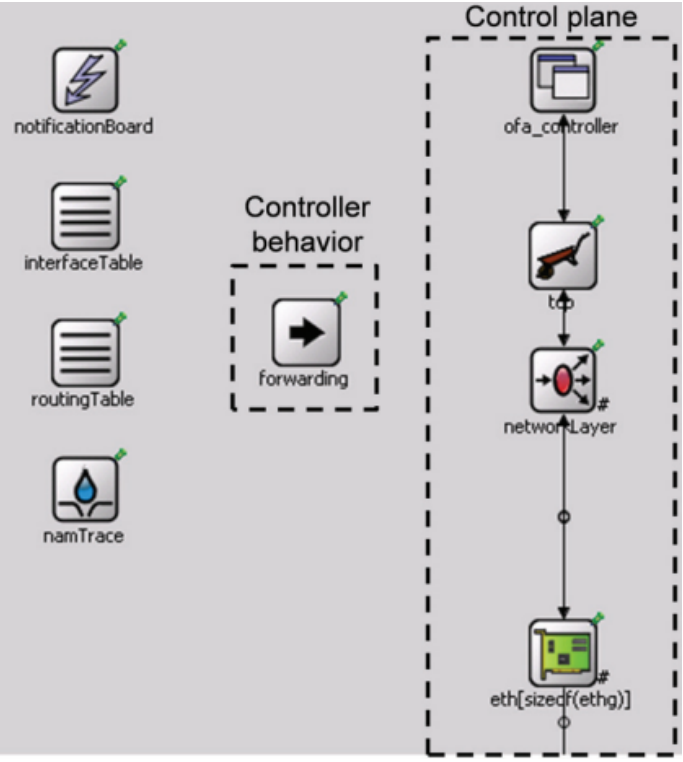
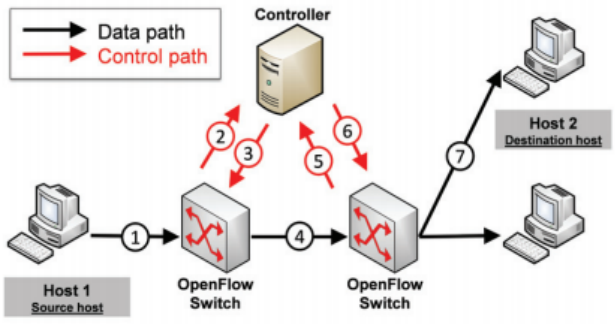


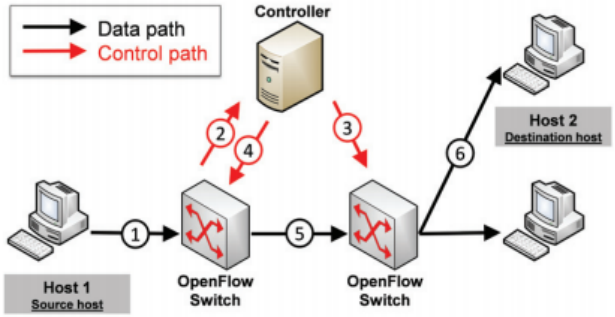
Figure 2.7: OfOMNeT Controller [50].

It, also, contains the $OFA\_Controller$ module that realizes the controller functionality. The $OFA\_Controller$ module contains a service time parameter which simulates the processing time of real OpenFlow controllers. The $OFA\_Controller$ module provides public methods which can be used to send OFP_Packet_Out and OFP_Flow_Mod messages the the OpenFlow switch. The controller behaviour is realized in a separate module called $OF\_Controller\_App$. This is a controller module interface that implements various controller behaviours. This way it is easier to configure the desired controller behaviour, just by altering a OFA_Switch configuration parameter. The packet-in signal are received from the $OFA_Controller$ module. Depending in the controller's behaviour set the received signal processes different operations and once finished triggers different public methods at the $OFA_Controller$ module. The authors implemented three operations for the controller behaviour the Hub, Switch and Forwarding modules.

- Hub and Switch behaviours: They model ordinary Ethernet and switch functionality, implemented as test for the OpenFlow protocol. For example in Figure 2.8(a),



(a) Switch behavior.



(b) Forwarding behavior.

Figure 2.8: OfOmnet Controller Behaviours [50].

host 1 wants to send a packet to host 2. The packet arrives at the first OpenFlow switch which does not have an entry to its $FlowTable$, consequently it sends a $PacketIn$ message to the OpenFlow controller, which in turn sends a $FlowMode$ message and then a $PacketOut$ message to the first OpenFlow Switch. Then, the packet arrives at the second OpenFlow Switch which also does not have an entry in its $FlowTable$, therefore the same process is repeated. Afterwards, the packet arrives at its final destination.

- Forwarding Behaviour: This model has a complete knowledge about the network, therefore it knows which switches are intermediate between a target and a destination. It uses this knowledge to setup a path between the transmitter and the receiver. For example in Figure 2.8(b), when the packet arrives at the first OpenFlow switch, the controller sends a $FlowMod$ message to the second OpenFlow switch to setup a path for the packet, then it sends a $FlowMod$ message to the first OpenFlow switch. Finally, the controller sends a $PacketOut$ message to the first switch, in order to send the first unmatched packet through the path. The second behaviour decreases the number of messages send from the controller, and when the packet arrives at the second OpenFlow switch, it is forwarded immediately to it final destination, thus decreasing delay.

Chapter 3

Approach

Farcast's goal is to improve the switches' forecasting accuracy, using SDN. This chapter shows the procedure that Farcast uses to help the switches' predictions be valid, then it dwells deeper into the changes that need to be made to the controllers and switches in order for Farcast to be operational. Farcast's logic could be generalized to include all the switches in a network.

In order to test Farcast we designed a network, which is shown in Figure 3.1. This network consists of two computer networks (Network 1 and Network 3) multiple OpenFlow switches and a controller. Network 1 forwards traffic to Network 2, through three access switches, then through various topologies of OpenFlow switches to a prediction switch. Afterwards, the prediction switch forwards the traffic to Network 2. The prediction switch makes predictions about the amount of traffic it will forward to Network 2. The controller is responsible for making the prediction switch accurate in case it detects an anomaly. In

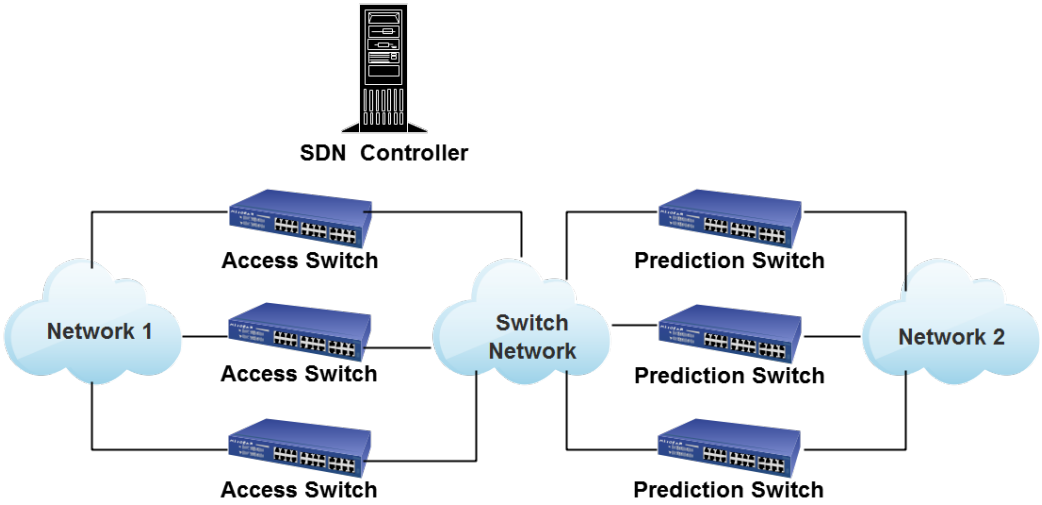


Figure 3.1: A network using Farcast.

case the controller estimates that the flows in the network are not sufficient for a switch to achieve it prediction, it has to reroute the flows in the network, accordingly. Section 3.1 contains a simple example of good Farcast functionality and decisions.

3.1 A Simple Use Case

In this section, a simple case of Farcast functionality is demonstrated by using a simple example. Figure 3.2 shows a simple network, in which a client sends traffic to a server. In between the client and the server are 4 switches. The switches forward the traffic towards the server through the routes established by the SDN controller. The number of different paths from the server to the controller is 2, while the switches that make the predictions are the 2 parallel switches in the middle.
For simplicity, we assume that there exist only two flows within the network that have the

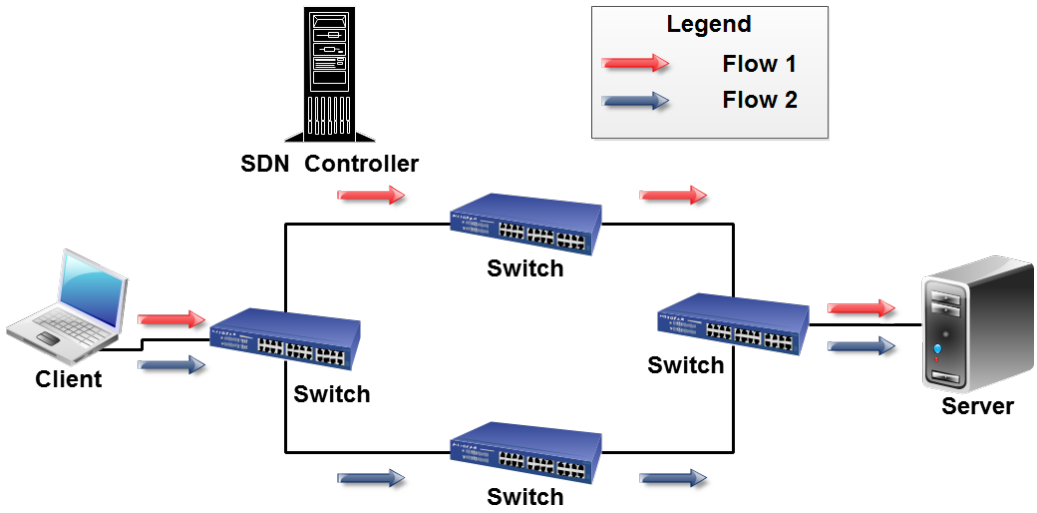


Figure 3.2: 2 flows in the network.

same data-rate. These are Flow 1 (red) and Flow 2 (blue) and each one follows a different path to reach the server. Naturally, the each one of the two switches predicts an x amount of traffic as the two flows have the same data-rate.

Figure 3.3 shows a change in the state of the flows of the network. Now Flow 2 (blue) is removed from the network, while Flow 1 remains unchanged. Furthermore, a new flow, named Flow 3 (purple) is inserted in the same path as Flow 1 (red). For simplicity Flow 3 has the same data-rate as Flow 1.

This scenario makes both of the predictions go wrong, as the switch in the path of Flow 1 has predicted an x amount of traffic but it the traffic amount will, actually, be $2x$ because of the new flow inserted in the path. Additionally, the switch below has predicted an x

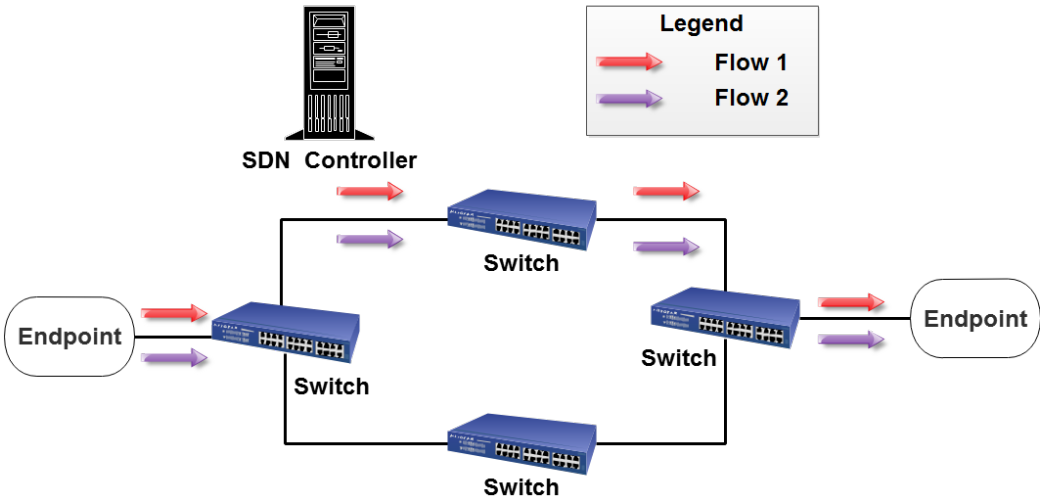


Figure 3.3: Flow 2 down and flow 3 inserted.

amount of traffic, while there is no traffic traversing through its path. The controller is able to fix these predictions by stitching the path of either Flow 1 or Flow 3 to be the one previously followed by Flow 2. This is shown in Figure 3.4, where Flow 3 (purple) is reassigned to the s path. Now, both switches will achieve their predictions of x. In

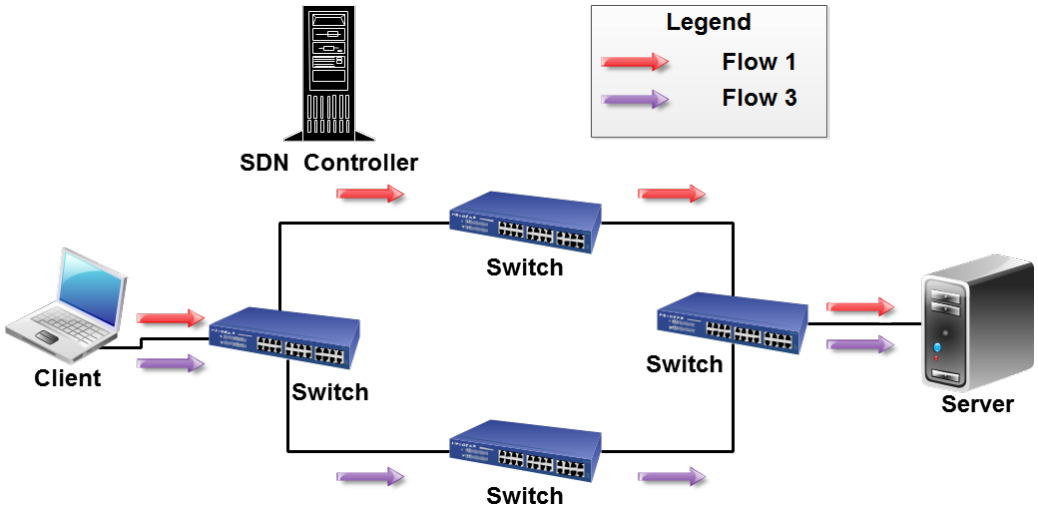


Figure 3.4: Re-routing to aid forecasting.

this example, the accuracy of the prediction will be affected by the reaction time of the controller, meaning that the sooner the controller realises there is something wrong with the prediction of the switches, the sooner it will be able to stitch the paths and help them in their predictions. In the following sections, we show in depth, how Farcast works to achieve this functionality.

3.2 Prediction Intervals

The controller has to be in close communication with the switches, in order to be informed of the switches' predictions and the flows that traverse through the network. Figure 3.5 shows the time slots in which the controller-switch communications take place. Here, one prediction interval is shown. This prediction interval is divided into several inform intervals. The prediction intervals denote the times that the switches make predictions for the future traffic, while the inform intervals denote the time slots, where the switches inform the controller of the flows that traverse the network and the total amount of traffic that was forwarded outside the network up to this point. Thus, the prediction interval is the time between two consecutive predictions, while the inform interval is the time between two consecutive information slots. In Figure 3.5 the prediction interval is 60 seconds, while the inform interval is 20 seconds.

Each prediction corresponds to the traffic amount that the switch expects to forward

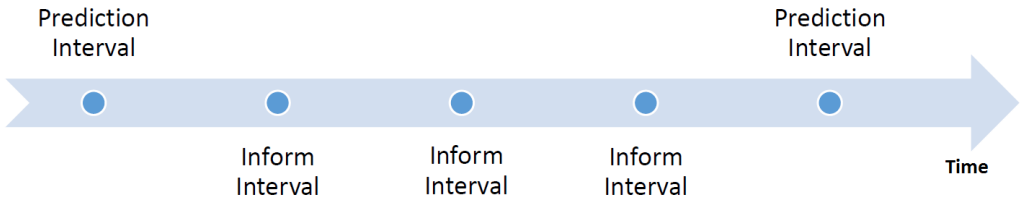


Figure 3.5: One Farcast Prediction interval.

until the next prediction. Thus, in Figure 3.5 the prediction made in the first prediction interval corresponds to the traffic load expected until the 60^{th} second. This prediction is sent to the controller immediately.

The information sent by the switches to the controller in the inform slots, correspond to the traffic that traversed the network in the previous information interval. Thus, in Figure 3.5 the information sent to the controller at the 40^{th} second corresponds to the flows and traffic between the 20^{th} and the 40^{th} second.

Farcast interval's logic is summarized in Figure 3.6. In each prediction phase the prediction switches make their predictions and send them to the controller. In each inform phase, the switches send to the controller information about the flows. Then, the controller makes the necessary corrections to the flows paths, if needed, in the correction phase. This loop is continued until the end of the prediction phase. Finally, the process starts all over at the end of the prediction interval, where the switches send their new predictions to the controller. When the controller receives the information about the traffic, it assesses whether the prediction made at the start of the prediction interval is correct (more on that in section 3.5). Afterwards, it reroutes the flows if needed to help the switch achieve its prediction. This procedure is followed in each inform interval.

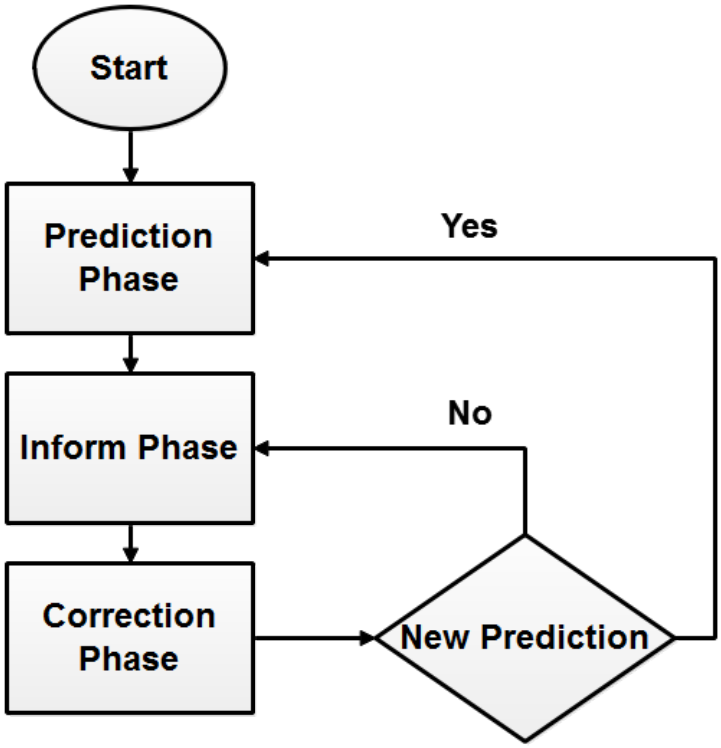


Figure 3.6: Farcast phases.

3.3 Farcast Flowchart

Figure 3.7 shows the flowchart of the controller's logic in Farcast. The initial state, denoted with the double circle, is when a new prediction arrives at the controller. The controller saves this prediction and goes to the next state where it checks whether the prediction interval has ended. As we are following the example of Figure 3.5 and the prediction has just arrived, it goes to the next step which is checking whether the Inform interval has started. The controller switches back and forth, between these two steps named the intervals ending check loop, until the Inform Interval Start occurs. When that happens, the controller receives the flow updates from the switches. Having this information, the controller can now assess whether the switch's prediction is accurate. If the prediction is accurate or if the prediction is not accurate but the controller does not have the necessary flows to help the switches, the controller returns to the intervals ending check loop. Otherwise the controller reroutes the suitable flows and returns to the intervals ending check loop. When the prediction interval reaches its end, the controller reroutes the flows it has already rerouted in this prediction interval, because if it did not

reroute them, these flows would make the next switch prediction inaccurate. Additionally, the controller sends a message to the switch to inform it about the amount of help it gave it to reach the predicted traffic amount. Otherwise, the switch would assume that it was right about its prediction and continue making wrong predictions until the link becomes empty or overloaded.

In the rest of this chapter, we discuss in depth about the switches and the controller's

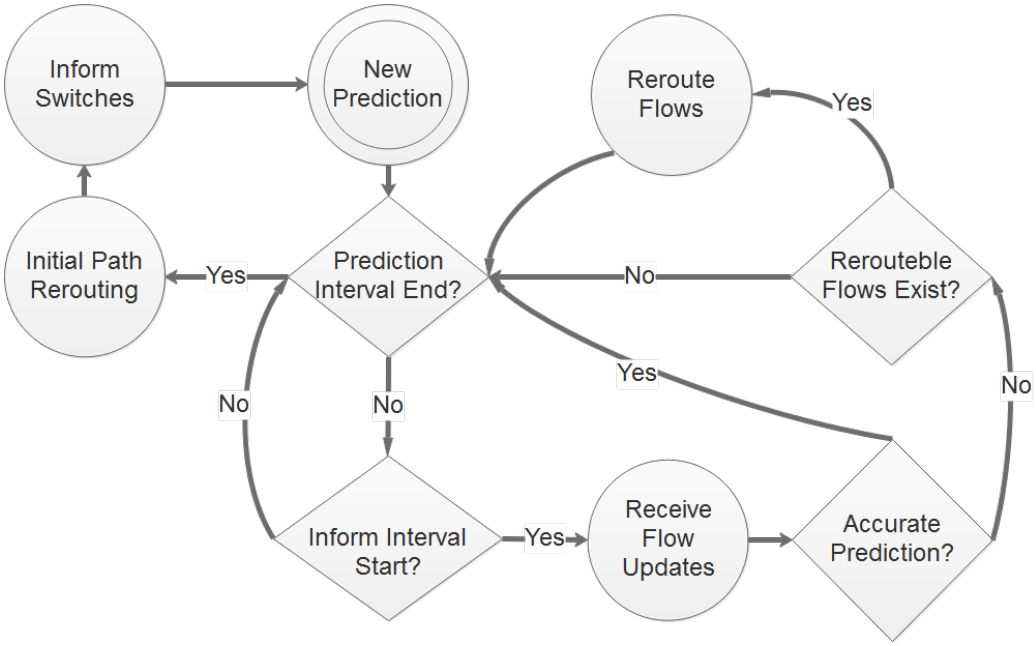


Figure 3.7: Farcast logic flowchart.

behaviour, the Farcast's limitations and the changes made to OfOMNeT to support Farcast.

3.4 Switch Functionality

In Farcast, the OpenFlow switch functionality is extended, as the switches at the end of the network predict the future amount of traffic, forwarded outside the network. This is the first alteration of the OpenFlow switches functionality, however this alteration triggers other changes that need to be made in order for Farcast to work as expected.

Initially, the OpenFlow switches sample at each time interval called prediction interval as mentioned in section 3.2, the amount of traffic they forward outside of the network. The prediction intervals are set at the initialization of the OpenFlow switches and do not change afterwards. The prediction interval times do not change, since the time-series forecast algorithms require samples at the same time intervals to predict the amount of

traffic of the next interval. After gaining a satisfying number of samples, the switches start making predictions on the future amount of traffic. The prediction is made using the ARIMA time-series forecast algorithm, section 2.2, then it is sent to the controller. The controller ensures that the switch's prediction will come true.

There are two cases, where the switch may make a wrong prediction. First, if it predicts a higher amount of traffic than the real amount of traffic and second, if it predicts a lower amount of traffic than the real. In both cases, the controller has to make sure that the switch learns that it made a wrong prediction. Otherwise, if the switch believes that it makes good predictions by seeing its predictions are accurate, it will continue making wrong predictions, as it will continue to predict higher and higher or lower and lower amounts of traffic, respectively. However, the switch needs to know the amount of help it received from the controller and subtract it from the amount of traffic it forwarded. Only then, the switch will continue making accurate predictions. Consequently, the switch must receive a message from the controller that informs it about the amount of traffic it received from the controller, at some point before the next prediction.

Additionally, the controller has to have knowledge of the network's traffic characteristics, in order to help the switch's prediction come true. Specifically, it needs to know each flow's bit rate, as, ultimately, the controller will alter the paths of specific flows to add or subtract traffic from a specific prediction switch. A simple approach to the problem of informing the controller about each flow's rate would be the switches at the entrance of the network to inform the controller about the flows they forward, just before the controller is about to make an assessment on whether the prediction switches are right or wrong, more on that in section 3.5. The intervals in which the controller makes these assessments are called inform intervals, as already mentioned in 3.2. In order to make an approximation of each flow's rate, the switches count the number of bytes of each flow from the first encounter of the flow inside an inform interval, until the end of the interval, afterwards the total number of bytes of the flow is divided by the time it is present in the switch.

Finally, the controller needs to know each prediction switch's amount of traffic forwarded at each inform interval. The controller receives the flow rates from the switches at the network's entrance, as well as, the amount of traffic forwarded to the server, at each inform interval. Only then, the controller will have enough knowledge to make an assessment whether the prediction for each switch was right or wrong in each inform interval.

3.5 Controller Functionality

In Farcast, the controller has to implement additional functionality apart from its OpenFlow feoatures. Its job is to make sure that the switches achieve their predictions regarding the future traffic they forward outside of the network. As mentioned in section 3.4, the controller receives messages from both the switches at the entry and at the exit of the network, in order to be informed for the flows that exist in the network and the predic-

tions of the switches, respectively. These messages are exchanged after each inform and prediction interval, respectively. For simplicity, assume that the controller monitors only one switch.

After each inform interval the controller has knowledge of both the switch's predictions and of the flows that traverse the network and their bit rate. Then, the controller needs to make an assessment of whether the switch has made a right prediction or not by making its own prediction for the future traffic of the switch. The formula for the controller prediction in each interval is given in equation 3.1.

$$P = \sum_{x=1}^{N} f(x) T_{next} \tag{3.1}$$

Where:

- P is the controller's prediction about the switch's amount of traffic.
- N is the number of flows assigned to the given switch.
- T_{next} is the time that remains until the next prediction interval.
- $f(x)$ is the bit rate of the x^{th} flow.

After making its own prediction, the controller checks if the controller's prediction is close to the switch's prediction, equation 3.2, if not the controller needs to reassign flows in order to make the switch achieve its prediction.

$$(1 - p)S \leq P \leq (1 + p)S \tag{3.2}$$

Where:

- S is the switch's prediction.
- P is the controller's prediction.
- p is a pre-set variable and $0 < p < 1$.

Variable p is used to adjust the percentage of prediction error that is tolerable in Farcast. If P is not within these bounds, the controller starts adjusting flow paths to achieve the switch's prediction. Say that D denotes the difference between the switch's prediction and the controller's prediction, equation 3.3, and P does not satisfy the equation 3.2.

$$D = S - P \tag{3.3}$$

Then, the controller has to add or remove flows from the switch to achieve the switch's prediction. The controller has available a number of flows that can be rerouted. The next step is the calculation of which is the best combination of flows that needs to be rerouted, to achieve the switch's prediction. This is known as the knapsack problem. A simple

solution would be to sort the flows by data rate then start adding/removing flows until no more flows are needed to be added/removed [52].

Thus, the controller decides which flow's paths to modify by sorting the flows starting from the flow with the smaller bit rate, then adding or removing flows using the equation 3.4.

$$X = \begin{cases} \sum_{x=1, x \notin A}^{k} f(x) T_{next}, & \text{if } D > 0 \\ \sum_{x=1, x \in A}^{k} f(x) T_{next}, & \text{otherwise} \end{cases} \tag{3.4}$$

Where:

- X denotes the sum of the controller's help to the switch. X cannot be greater than $|D|$.
- A is the set of flows assigned to the switch.
- k is the last flow for which the condition $X < |D|$ stands, after the flows are sorted.
- T_{next} is the time that remains until the next prediction interval.
- $f(x)$ is the bit rate of the x^{th} flow.

This procedure summarizes the help given by the controller to the switch it monitors. Additionally, the controller makes sure that the switch is not misled into believing it has made right predictions, as it would affect its next predictions. Consequently, the controller reroutes the changed flows at the end of the prediction interval and informs the switch about the amount of help it received from the controller. This ensures the correctness of the next switch's predictions.

3.6 Optimal Calculation of Path Load

In this section Farcast is generalized to monitor all the prediction switches at the end of the network. This stage receives as inputs the current traffic matrix and a set of possible paths connecting each node pair in the network. It then produces the optimal aggregate load that each network path should carry. This problem can be formulated as a Linear Program (LP) as follows.

Let $M_{e,e'}$ be the current traffic matrix containing the data rates between any two nodes e and e'. Let the triplet $\langle e, e', k \rangle$, $k = 1 \dots K$ index each of the K-paths that can connect the entities e, e'. Furthermore, let $f_{\langle e,e',k \rangle}$ represent the fraction of $M_{e,e'}$ over path $\langle e, e', k \rangle$. The objective of stage 1 is then achieved as follows [**?**]:

$$\begin{array}{cc} \text{minimize:} & \mathcal{U} \\ \text{subject to:} & \\ \forall_{e,e'} : & \sum_{\forall k} f_{\langle e,e',k \rangle} = 1 \\ \forall \langle e,e',k \rangle, \forall l \in \langle e,e',k \rangle: & \sum_{\forall e,e'} M_{e,e'} f_{\langle e,e',k \rangle} \leq \mathcal{U} \cdot C_l \end{array} \tag{3.5}$$

where C_l is the nominal capacity of link l and $\mathcal{U} \in [0, 1]$ a helper variable. The first condition expresses the conservation of the traffic load, while the second one ensures that the load of each link is within its capacity constraint.

In the farcast case, we monitor, forecast and regulate the traffic over a set of network links $\mathcal{L}$. At each minor time-tick, we derive a traffic rate that each link should carry in order to meet the forecasting quotas. At a given time-tick, let these traffic rate values be:

$$\rho_l \cdot C_l,\ l \in \mathcal{L},\ \rho_l \in [0, 1] \tag{3.6}$$

These required traffic rate values can be treated as extra restrictions. In other words, the second restriction of eq. (3.5) now takes the following form:

$$\begin{array}{c} \forall \langle e,e',k\rangle, \forall l \in \langle e,e',k\rangle, l \notin \mathcal{L}\text{:} \sum_{\forall e,e'} M_{e,e'} f_{\langle e,e',k\rangle} \leq \mathcal{U} \cdot C_l \\ \forall l \in \mathcal{L} : \sum_{\forall e,e'} M_{e,e'} f_{\langle e,e',k\rangle} = \rho_l \cdot C_l \end{array} \tag{3.7}$$

Equation (3.5) can now be solved with any Linear Program solver, deriving the $f_{\langle e,e',k\rangle}$ values, i.e., the aggregate traffic that each network path should carry.

There is a case where eq. (3.5) may be insolvable, since there may be insufficient traffic in the network to support the data rates of relation (3.6). In this case, the administrator must first prioritize the links in the $\mathcal{L}$ set. The link with the least priority is removed from the $\mathcal{L}$ set, and we attempt to solve eq. (3.5) anew. If the equation remains insolvable, we proceed to remove the least prioritized link from $\mathcal{L}$ iteratively. If the set $\mathcal{L}$ becomes empty, we conclude that the farcast process cannot run in the given network conditions.

Mapping of entity pairs to paths. Given the current traffic matrix $M_{e,f}$, the current flow-to-path map and the current flow rates, this stage maps flows $\langle e, e'\rangle$ to paths $\langle e, e', k\rangle$ in order to match the optimal $f_{\langle e,e',k\rangle}$ values produced in stage 1. We proceed to update the Farcasting logistics, as well as keep an updated track of the original path of each moved flow (i.e., as it was at major time-ticks), exactly as in the one-link farcasting case. The mapping is heuristic each time, we seek to move the least number of flows from their paths, in order to meet the optimal $f_{\langle e,e',k\rangle}$ values produced in stage 1, using equation 3.4.

3.7 Farcast Limitations

In order for Farcast to function as expected, the following cases should be taken into consideration.

First, if the switch has made a prediction and the amount of traffic it expects is lower than the actual amount of traffic assigned to it, the controller will assign a number of flows to the switch to make up for this wrong prediction at some inform interval. However, since the internet flows are bursty, the flows could terminate after the controller assigns them to the switch and before the next inform interval. This means that the controller has helped

the switch less than the controller expected. Moreover, at the start of the next prediction interval the controller will give false information to the switch, regarding the amount of help that the switch received from the controller. As a result, the switch will have stored wrong measurements regarding the amount of traffic that flows through it. This scenario affects negatively the next prediction of the switch.

This problem can be mitigated by altering the paths of the flows that have large durations. This scenario is feasible, since it is observed that in the Internet there are dragonflies and tortoises [53, 54, 55, 56], meaning there exist many short-lived and a smaller number of long-lived flows. Specifically, there is a 20% percentage of flows that have larger duration than the typical short-lived flows, some of these long-lived flows have a lifespan that ranges from 10 minutes to even weeks. Examples of long-lived flows are the flows that are associated with video/audio applications, torrents, games or network monitoring tools. It is also important to note that the flow duration is not necessarily associated with the flow size.

Another argument that supports this scenario is that there has been extensive work over the years on flow classification [57], either port forwarding based [58, 59, 60] which is less reliable than the packet payload inspection for specific string patterns of known applications [61, 58, 62, 63, 59]. Furthermore, additional work has shown that even the packet payload inspection methods are not necessary, as flow classification can be realised by taking advantage of the social interaction of hosts [62, 64, 60] or flow feature extraction, such as flow duration, number and size of packets and interarrival times, [65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75].

These arguments indicate that Farcast can become more reliable by altering the paths of the flows that have the largest lifespans, because it is less likely for these flows to terminate in an information interval. The controller could distinguish the long-lived flows using the flow classification techniques mentioned above and then stitch their paths accordingly. The implementation of the flow classification methods is out of the scope of this thesis and will not be further discussed. However, the fraction of long lived/short lived flows is something that requires further testing.

Second, in case the switch has made a prediction and the amount of traffic it expects is higher than the actual amount of traffic assigned to it, the controller will assign a number of flows from the switch to different switches to make up for the wrong prediction. Since, the internet flows are bursty, the flows could be terminated after the controller assigns them to the other switches and before the next inform interval. As in the first case the controller has helped the switch less that it expected, meaning it will give false information to the switch and affect the switch's next prediction, negatively. Again this problem can be mitigated by altering the paths of the most long-lived flows.

Third, as mentioned in 3.4, after each inform interval, the access switches send the controller information regarding the current flows. Also, the server switch informs the con-

troller regarding the total amount of traffic it forwarded, so far, towards the server. Then, the controller makes an estimation to assess whether the switch is right about its prediction or not (see section 3.5). If the controller decides that the switch is wrong, it alternates the paths of the flows, that if they are removed from/assigned to the switch can make the prediction true. Then, the controller sends the FlowMod messages to the switches to stitch the new paths. The controller's reaction time, Ts, is the time it takes the controller to follow the procedure described above. If the controller's reaction time is not faster than the inform interval time, the controller will not be able to help the switch, since the controller will not be able to affect much the traffic that traverses the switch in each inform interval. This problem could be solved by taking measurements of the controller's reaction time and ensuring that it is always significantly smaller than the information interval. The relation of the controller's reaction time and the information interval could be found experimentally.

Fourth, if the unexpected traffic behaviour takes place at the last inform interval, the controller will be unaware of the traffic state and will not be able to help the switch. This blind spot does not affect much Farcast's performance, since the blind spot's time duration can be decreased by choosing the inform intervals appropriately. For example if the prediction interval is 60 seconds while the inform interval is 10 seconds, then then blind spot time duration is 10 seconds, meaning $\frac{1}{6}^{th}$ of the prediction interval. The inform interval can be decreased to further decrease this fraction of time, however care should be taken as calibration is needed to find the balance between the third and the fourth limitation.

Finally, the controller should always take care that the total throughput in each available path does not exceed the path's maximum capacity. This problem can be solved just by the controller knowing each path's limit and taking care not to exceed it and will not be further discussed in this thesis.

Chapter 4

Evaluation

4.1 Benchmark

Farcast was implemented in OfOmnet (for a description of OfOMNeT, refer to section 2.6). Our aim was to simulate a network consisting of a set of OpenFlow switches that forward the Internet traffic to the outside world. To accomplish this aim, a network was developed, using the NED language, supported by the OMNeT++. The scenario is shown in Figure 4.1.

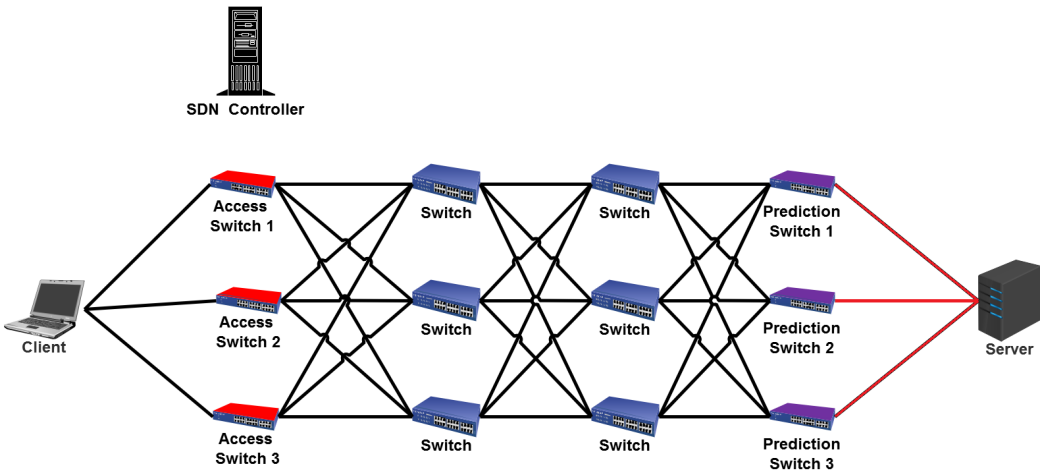


Figure 4.1: Farcast test scenario.

In this scenario, the client, shown in Figure 4.1, forwards the packets of each UDP [76] flow towards the server. The server is only responsible for consuming the data sent by the client, it does not send acknowledgements to the client. The only response that the server has to send back to the client is the ARP reply as specified by the ARP protocol [77]. We chose this approach, since the UDP protocol is more predictable than the TCP [78], as TCP operates using the mechanism known as congestion control. Furthermore,

we wanted to avoid the overhead of the server acknowledgements in our simulations.

Figure 4.1 shows one access switch that forwards to client's packets into the network and one server switch that forwards the packets from the network to the server. The network is consisted of 4 layers of switches. Each layer consists of 3 switches. Each switch in the network is connected to all the switches of the next and previous levels. The controller is connected to all the switches and is responsible for stitching each flow's path to its final destination, the server. Each link's capacity is 100 Mbps.

In section 3.5, the controller's functionality is described. In this paragraph we describe, in short, the changes that were introduced in OfOMNeT to accomplish this functionality. The controller module has to be able to monitor the current network's state in each prediction and inform interval (for a description of the intervals, refer to the introductory parts of chapter 3). The state of the network that the controller has to have knowledge of, consists of each flow's characteristics (port destination and rate in MB/s) and each OpenFlow switch's prediction (only the switches that connect to the server switch make predictions for the future traffic). Therefore, the controller module under Farcast is able to receive information from the switches regarding the state of the network, then it can make an assessment whether each switch that made a prediction is right or wrong. Moreover, an algorithm to address the knapsack problem was introduced, as described in section 3.5. Additionally, a depth-first-search algorithm (DFS) [52] was implemented, in order for the controller to find all the paths from the switch, that sends the PacketIn to the controller, to server. The DFS finds all possible paths, however the controller chooses only the shortest and link disjoint paths. Since, there are only three OpenFlow switches at each side of the network, the controller chooses, always, three unique shortest disjoint paths. Only these paths are used for the path stitching. Finally, the controller is able to send information about how much help each switch received to the switches that made the predictions.

As stated in section 3.4, in each predict interval, the OpenFlow switches that connect directly with the server switch (purple switches in Figure 4.1) need to record the amount of traffic sent to the server switch. Using these recordings they call the ARIMA algorithm, implemented in Python, in order to predict the amount of traffic at the next prediction interval. Following the prediction, the OpenFlow switches need to send the prediction to the controller, as the controller is responsible for monitoring the network and helping the OpenFlow switches achieve their prediction in each interval. Furthermore, in each predict interval, the OpenFlow switches that are directly connected with the server switch have to accept an inform message from the controller. The inform message contains the amount of help received by the controller. The amount of help received by the controller is then subtracted from the amount of traffic that passed from the OpenFlow switch to the server switch, since the real traffic if not for the controller's help, would be the sum of the controller's help and the traffic that went through the switch. Additionally, the OpenFlow switches that directly connect with the access switch, meaning the entrance to the network (red switches in Figure 4.1), need to send each flow's rate to the controller, to assist the controller's estimation of the predictions success or failure.

The number of extra messages used by Farcast to monitor the network are given by the following equation:

$$N = SI + P(I + 2) \tag{4.1}$$

Where:

- N is the number of extra monitoring messages.
- P is the number of monitored prediction switches.
- I is the number of inform intervals.
- S is the number of access switches that inform the controller about the flows.

4.2 ARIMA Implementation

In this work we used the ARIMA model implemented in the class statsmodels [79] for python. The ARIMA model was trained on real-world data obtained from the Department of Information Systems [80]. We trained the ARIMA on real world data, because the objective of this thesis is to improve a forecasting method's prediction validity, using SDN. Therefore the training of the machine learning algorithm for each simulation was out of the scope of this thesis and too time consuming for our simulations.

The ARIMA method was called by the monitored switches at each time interval of 1 minutes, and provided the prediction of the traffic amount of traffic for the next time interval. The inform intervals lasted for 10 seconds each.

4.3 One Switch Results

In this section, the performance of Farcast is examined if only one switch is monitored. The controller receives the predictions of one switch and then reroutes the flows, accordingly, to make the monitored switch achieve its prediction.

The network forwards 30 UDP flows from the client to the server and the controller at first distributes these flows to the three disjoint paths, one of which contains the monitored switch. Afterwards, an elephant flow, meaning a flow that equals 10 UDP flows is inserted to each inform interval for the first experiment and removed for the second experiment, to assess the accuracy of the predictions with and without Farcast.

In Figure 4.2 an elephant flow is inserted to the network at the monitored switch's path, at each inform interval. This experiment was repeated 50 times for each inform interval, meaning that in order to provide results, 250 simulations were run. The MAPE shown in Figure 4.2 is the average MAPE of 50 times of simulation for each inform interval.

We expect that the accuracy of the ARIMA predictor to fall, since it predicts much lower traffic than the real, that actually traverses through the network.

Moreover, we expect Farcast to remove flows from the monitored switches path and be closer to the initial prediction of the switch. Since, the implementation of the flow re-routing is heuristic, we expect that the number of the transmitted messages, M, that the controller uses to reroute the flows is:

$$M = 2 * K * P \tag{4.2}$$

Where:

- M is the number of transmitted messages.
- P is each path's length, 5 in this test's case. The final switch does not count, as it always forwards the traffic towards the server and the controller does not need to send it re-routing messages.
- K is the number of flows that need to be rerouted, which is 10 in this test's case, since the flows that need to be rerouted are the 10 smaller flows that already exist in the monitored switch's path, before the elephant is inserted into this path.
- The number of transmitted messages are multiplied by 2, since the flows need to be re-routed at their initial paths, at the end of the prediction interval.

Figure 4.2 shows the MAPE, when inserting an elephant into the monitored switch's path, in random moments within each inform interval. The blue line corresponds to the ARIMA predictor without the controller's help. The MAPE in this case is at 200%, if the elephant is inserted in the first inform interval and falls to 40% if the elephant is inserted at the final inform interval. This is expected, since the sooner the elephant is inserted to the path after the prediction, the more erroneous the prediction will be, as more data will be transmitted within the prediction interval.

The red line corresponds to the Farcast's MAPE, which is significantly lower than the ARIMA predictor alone and is always stable at 20%. Therefore, Farcast can help the switch come closer to achieving, its prediction. In this case, using 100 re-routing messages for removing flows from the monitored switch to the other switches, plus the 26 messages used for monitoring the network.

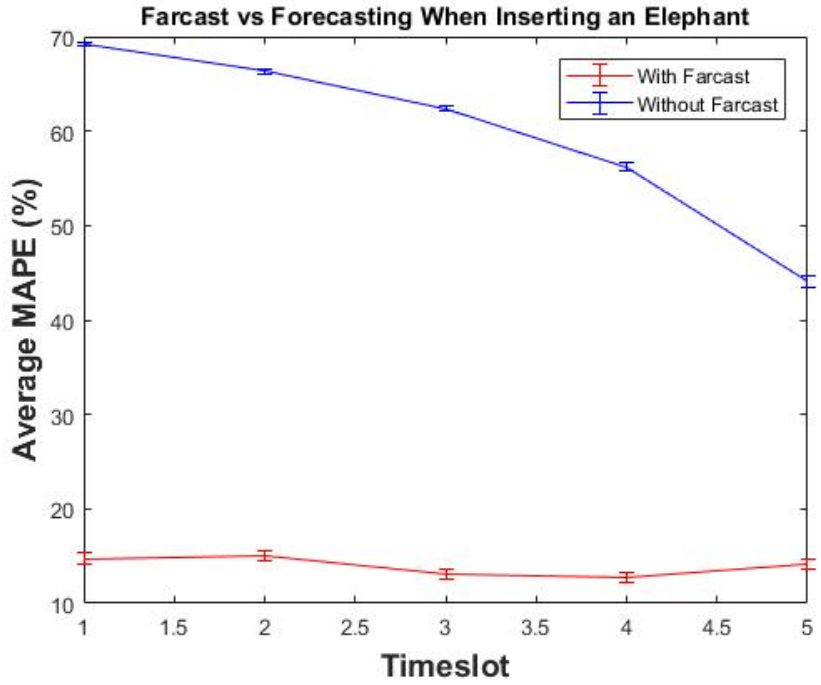


Figure 4.2: Inserting an elephant flow in each inform interval.

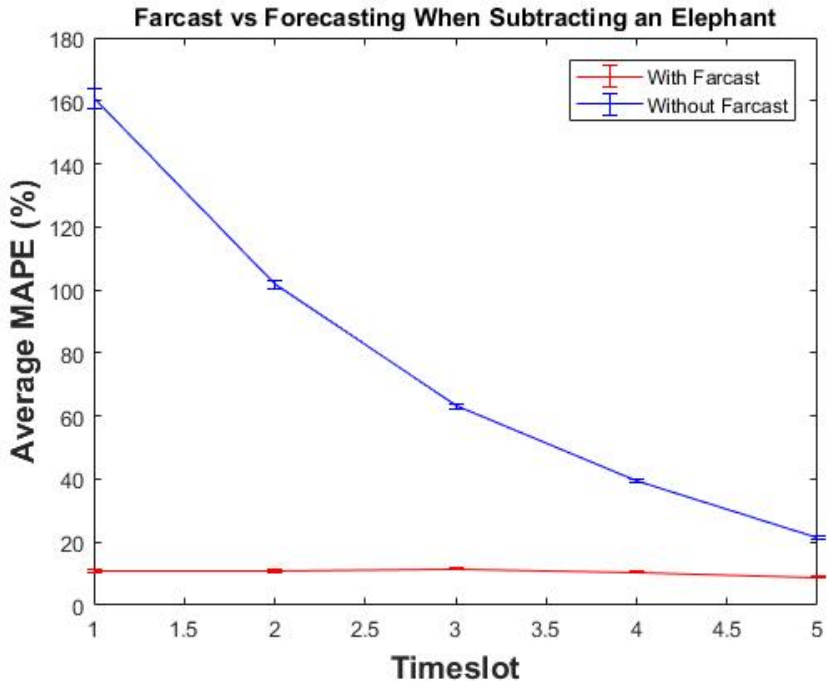


Figure 4.3: Subtracting an elephant flow in each time slot.

In the second experiment 30 flows were shared across the three paths, however this time an elephant was inserted to the monitored switch's path at the start of the simulation. Therefore, the ARIMA predictor expects and predicts the traffic of both the 10 mice flows and the 1 elephant's flow for the monitored switch.

Figure 4.3 shows the performance of an ARIMA predictor in comparison to the performance of the Farcast predictor, when an elephant is subtracted from the monitored switch's path in a random moment within each inform interval. The experiment was repeated 50 times for each inform interval.

The blue line corresponds to the ARIMA MAPE and shows the accuracy of a predictor of an unexpected event without using Farcast. The MAPE starts at 52% if the elephant is subtracted at the first inform interval and gradually falls for each inform interval, until it reaches 11% at the final inform interval.

The red line shows the Farcast's performance in this case, which is again lower than the ARIMA predictor's alone and stable around 9%. Ten flows are removed by the controller from the other switches to make the monitored switch achieve its prediction, therefore the number of exchanged messages is 100 for the re-routing, plus 26 for the monitoring. The MAPE here is lower than the MAPE in the previous experiment, since the MAPE metric favours predictions that are too low.

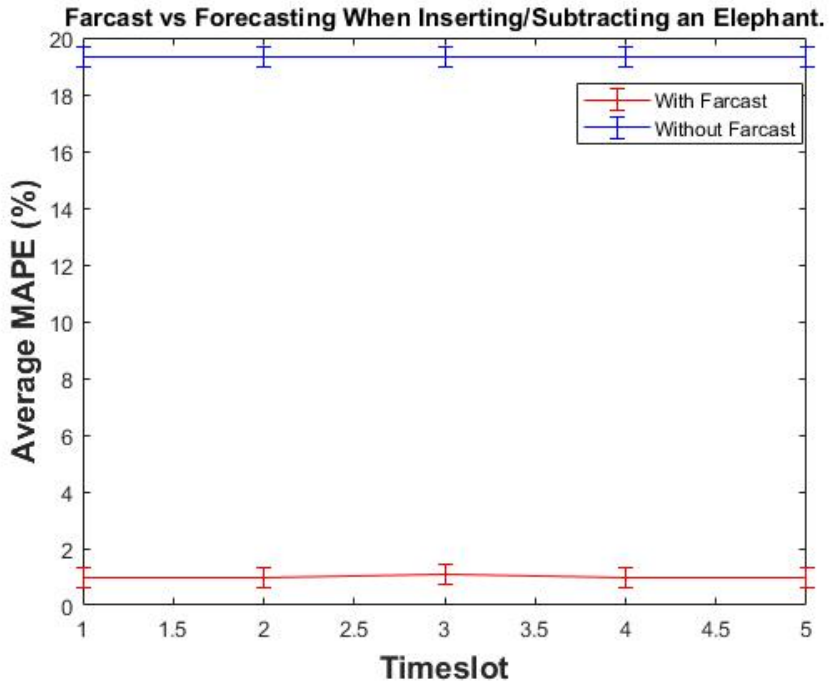


Figure 4.4: Inserting/Subtracting an elephant flow in each time slot.

In the third experiment each path is given 10 flows at first. Then, an elephant is inserted at a random moment in each inform interval, finally it is removed from the network at the inform intervals end. This reflects a bursty behaviour of a network, where a large amount of traffic is inserted to the monitored switch's path for a small amount of time. Each simulation was repeated 50 times for each inform interval.

Figure 4.4 shows the MAPE of the ARIMA predictor and Farcast for this situation. The blue line corresponds to the ARIMA predictors accuracy, which is stable at 18% for each inform interval. This is expected, as we insert the same amount of traffic in each inform interval, therefore the MAPE should be the same, independent of the inform interval. The red line corresponds to Farcast's MAPE, which is significantly lower than the ARIMA predictor's alone, at 1%. The number of the exchanged messages is again 100 for the re-routing and 26 for the monitoring messages.

In each experiment, it is shown that for one monitored switch, Farcast lowers the MAPE significantly, therefore Farcast can be used for improving the predictor's performance. In the next section, the results of Farcast, when monitoring all the prediction switches at the end of the network are shown.

4.4 Three Switches Results

In this section, we study the performance of Farcast, when all the prediction switches are monitored by the controller. The main difference with the previous section is that now, the controller receives prediction messages from all the prediction switches and tries to minimize all of the predictions MAPE.

The limitation of Farcast is that the controller cannot create or terminate flows that already exist in the network. In order to minimize the MAPE error, the controller needs to reassign flows that already exist to the prediction switches path. Therefore, when the controller estimates that only one switch has made an erroneous prediction about the load of traffic it expects in the future, it has to make a decision. In Farcast, the prediction switches are prioritized, meaning that the controller tries to fix some predictions in expense of other predictions.

At the initialization stage, the controller sets a percentage of error that the network can tolerate for each switch. This is of course, a choice of the user that implements Farcast and can be set to zero for each switch. However, if this percentage is set to zero, then the controller will not be able to fix predictions for cases where only one prediction switch is wrong on its prediction. Do not confuse the percentage error that Farcast tolerates for each switch with the MAPE extracted after the simulation is finished.

Farcast works best in scenarios where the traffic is bursty for more than one prediction switch. Ideally, when one switch predicts lower traffic than the actual, while another pre-

dicts higher traffic than the actual. In these cases Farcast can reassign flows from the second switch to the first and minimize both switches' MAPE.

This is shown in Figure 4.5, where while the switch predicts higher traffic than the actual while the rest of the prediction switches forecast lower traffic than the actual, meaning that Farcast can make their erroneous predictions cancel out.

In Figure 4.5, switch 10 makes a prediction and then an elephant comes through its path in the next prediction interval. This sets its prediction MAPE error 65%. However, switch 11 and 12 predict an amount of traffic when suddenly 5 flows are removed from each one's paths. This event makes their MAPE error 32% and 32%, respectively. The controller intervenes and flows from the switch 10 are stitched to the other two paths making the MAPE errors go down to 14%, 10% and 17%, respectively. For this scenario, the

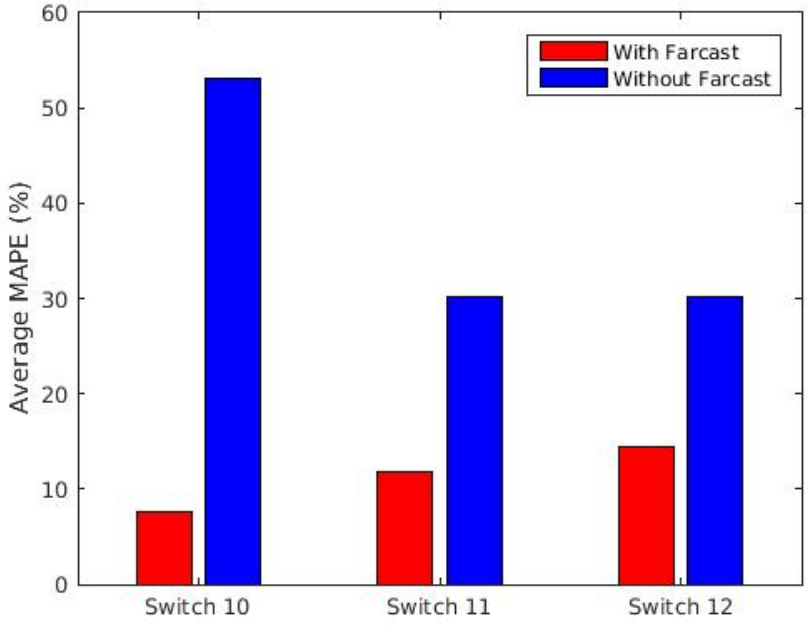


Figure 4.5: Inserting an elephant flow when monitoring all switches.

average number of transmitted messages is 136, meaning an average 100 messages for re-routing and 36 standard messages for monitoring. We know that the number of extra flows removed from switch 10 is 2 since the number of the re-routing messages is 120, and Farcast sends 10 re-routing messages for each re-assigned flow, 5 for the initial path stitching and 5 at the end of the prediction interval where the flows are rerouted to their initial paths.

The opposite scenario is shown in Figure 4.6, where this time, the elephant is removed from switch 10, while 5 flows are assigned to each of switch 11 and 12. The controller

assigns the extra flows of switch 11 and 12 to switch 10, lowering the MAPE error of all the switches.

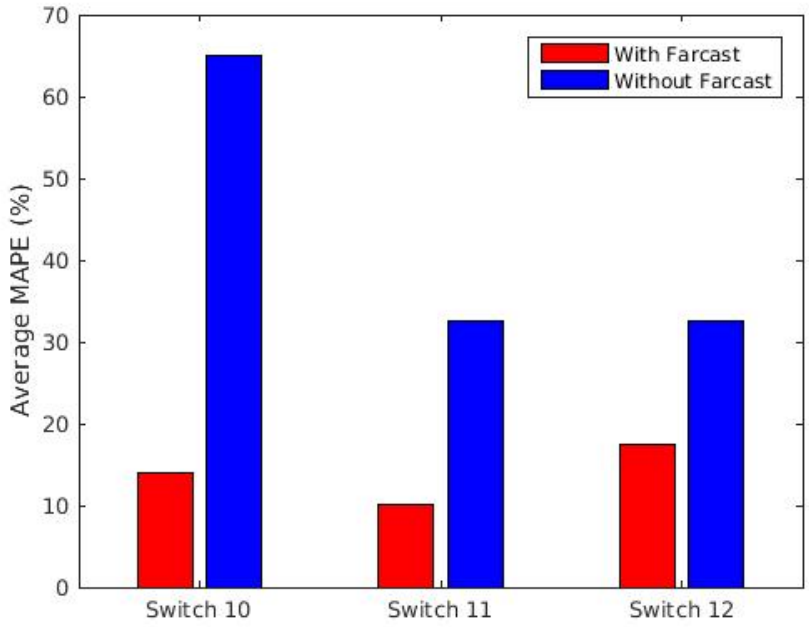


Figure 4.6: Subtracting an elephant flow when monitoring all switches.

In this case the MAPE error without using Farcast is 53%, 30% and 30% for switches 10, 11 and 12, respectively. The number of exchanged messages is 136 again, as the scenario is very similar to the previous one. In both simulations, the MAPE of switch 12 is always higher than the MAPE of switch 11, since switch 12's predictions has been set be of lower importance to the controller.

Chapter 5

Conclusions and Future Work

In this thesis, an approach named Farcast was designed and evaluated for improving forecasting accuracy via SDN. The methodology covers all steps from the moment the switches make the predictions about the amount of future traffic and send them to the controller, to the monitoring of the network's state and the re-routing of flows. In the experiments we examined how much Farcast can improve the forecasting accuracy in each time slot an anomaly happens. Moreover, we examined how much it can improve the forecasting accuracy, when three links are monitored and there is no backup link to accept extra traffic in case of a traffic burst.

Farcast showed that it is capable of improving the forecasting accuracy, in some instances as high as by one magnitude of order compared to simple forecasting. Additionally, we showed that the number of extra messages sent is a function of the path length and the number of monitored switches. Furthermore, Farcast improves the forecasting performance independently of the forecasting method used. Therefore, in a network monitored by Farcast, the forecasting method can be improved independently.

The next step could be implementing Farcast and evaluating its performance under heavy loads of traffic used in XG-PON networks, to assess whether it can be used for improving the XGPON's bandwidth assignment. In the XG-PON standard different ASNs should cooperate to take full advantage of Farcast. This means that the ASNs should be interconnected and connected to a central controller, which will be able to redirect the flows amongst them. However, further experimentation is needed to prove that Farcast will be useful in XG-PON networks, as Farcast must adapt in the very fast Internet speeds of XG-PON.

Another step should be experimenting with the prediction intervals and the inform intervals length and finding the best lengths for forecasting accuracy improvement. The inform interval length is very important for Farcast, as the shorter the length of the interval, the faster the controller will be able to determine and eliminate possible outliers. However, the length of the interval should be long enough for the controller to be able to

react to the outliers. Experimentation is needed in order to find the best prediction-inform intervals ratio in order to maximize the efficiency of Farcast.

Moreover, in the current version of Farcast, the network components clocks were considered to be synchronized, meaning that the monitoring messages were sent by the switches at the start of the inform intervals without a prior request from the controller. Farcast needs all the monitoring messages at the start of the inform interval before making reassignment decisions. In a real world scenario, the controller should request the switches' predictions and monitoring messages, since in the real world it is not safe to assume that all the network components are fully synchronized.

Finally, a method for determining which flows are long lived and which are not should be implemented and attached to the controller, in order for the controller to make better reassigning decisions. The long-lived flows are safer for improving the switches' forecasting accuracy, since they are more likely to continue to exist within an inform interval in which they are reassigned. The reassignment of long-lived flows will make Farcast more dependable.

Bibliography

[1] Huma Parveen and Rishi Srivastava, "Network Traffic Analysis and Prediction – A Literature Review," *International Journal of Research and Development in Applied Science and Engineering (IJRDASE)*, 2016.

[2] C. Liaskos, X. Dimitropoulos, and L. Tassiulas, "Backpressure on the backbone: A lightweight, non-intrusive traffic engineering approach," *IEEE Transactions on Network and Service Management*, vol. to appear, pp. 1–14, 2016.

[3] P. Cortez, M. Rio, M. Rocha, and P. Sousa, "Internet Traffic Forecasting using Neural Networks." IEEE, 2006, pp. 2635–2642. [Online]. Available: http://ieeexplore.ieee.org/document/1716452/

[4] C. Katris and S. Daskalaki, "Comparing forecasting approaches for Internet traffic," *Expert Systems with Applications*, vol. 42, no. 21, pp. 8172–8183, Nov. 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0957417415004315

[5] M. L. F. Miguel, M. C. Penna, J. C. Nievola, and M. E. Pellenz, "New models for long-term Internet traffic forecasting using artificial neural networks and flow based information." IEEE, Apr. 2012, pp. 1082–1088. [Online]. Available: http://ieeexplore.ieee.org/document/6212033/

[6] T. Otoshi, Y. Ohsita, M. Murata, Y. Takahashi, K. Ishibashi, and K. Shiomoto, "Traffic prediction for dynamic traffic engineering," *Computer Networks*, vol. 85, pp. 36–50, Jul. 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1389128615001565

[7] J.-C. Bolot and P. Hoschka, "Performance engineering of the World Wide Web: Application to dimensioning and cache design," *Computer Networks and ISDN Systems*, vol. 28, no. 7-11, pp. 1397–1405, May 1996. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/0169755296000736

[8] P. Cortez, M. Rio, P. Sousa, and M. Rocha, "Topology Aware Internet Traffic Forecasting Using Neural Networks," in *Artificial Neural Networks – ICANN 2007*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos,

D. Tygar, M. Y. Vardi, G. Weikum, J. M. de Sá, L. A. Alexandre, W. Duch, and D. Mandic, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, vol. 4669, pp. 445–454, dOI: 10.1007/978-3-540-74695-9_46. [Online]. Available: http://link.springer.com/10.1007/978-3-540-74695-9_46

[9] S. R. Talpur and T. Kechadi, "A forecasting model for data centre bandwidth utilisation," in *The SAI Intelligent Systems Conference (IntelliSys 2016), CentrEd at ExCel London, London, United Kingdom, 21-22 September 2016*, 2016.

[10] S. Chong, S.-q. Li, and J. Ghosh, "Ann based forecasting of vbr video tra c for dynamic bandwidth allocation in atm networks," 1995.

[11] M.-S. Han, "Simple and feasible dynamic bandwidth and polling allocation for XGPON." Global IT Research Institute (GIRI), Feb. 2014, pp. 298–304. [Online]. Available: http://ieeexplore.ieee.org/document/6779186/

[12] P. Sarigiannidis, D. Pliatsios, T. Zygiridis, and N. Kantartzis, "DAMA: A data mining forecasting DBA scheme for XG-PONs." IEEE, May 2016, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/7495169/

[13] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 219–230, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1030194.1015492

[14] B. Huifeng, Z. Xianlong, Z. Hongfeng, and Z. Likun, "Traffic-load prediction based on echo state network improved by bayesian theory in 10g-epon," *The Journal of China Universities of Posts and Telecommunications*, vol. 22, no. 2, pp. 69 – 73, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1005888515606410

[15] "OMNeT." [Online]. Available: https://omnetpp.org/doc/omnetpp/manual/#cha:overview

[16] "INET Framework." [Online]. Available: https://omnetpp.org/doc/inet/api-current/inet-manual-draft.pdf

[17] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*, fifth edition ed., ser. Wiley series in probability and statistics. Hoboken, New Jersey: John Wiley & Sons, Inc, 2016.

[18] Qilian Liang, "Ad hoc wireless network traffic-self-similarity and forecasting," *IEEE Communications Letters*, vol. 6, no. 7, pp. 297–299, Jul. 2002. [Online]. Available: http://ieeexplore.ieee.org/document/1018756/

[19] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, Feb. 1994. [Online]. Available: http://ieeexplore.ieee.org/document/282603/

[20] A. Sang and S.-q. Li, "A predictability analysis of network traffic," *Computer Networks*, vol. 39, no. 4, pp. 329–345, Jul. 2002. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1389128601003048

[21] R. Wolski, "Forecasting network performance to support dynamic scheduling using the network weather service." IEEE Comput. Soc, 1997, pp. 316–325. [Online]. Available: http://ieeexplore.ieee.org/document/626437/

[22] P. Cortez, M. Rio, M. Rocha, and P. Sousa, "Multi-scale Internet traffic forecasting using neural networks and time series methods," *Expert Systems*, pp. no–no, Dec. 2010. [Online]. Available: http://doi.wiley.com/10.1111/j.1468-0394.2010.00568.x

[23] V. Alarcon-Aquino and J. Barria, "Multiresolution FIR neural-network-based learning algorithm applied to network traffic prediction," *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 36, no. 2, pp. 208–220, Mar. 2006. [Online]. Available: http://ieeexplore.ieee.org/document/1624547/

[24] L. Tang, S. Du, and S. Ji, "Forecasting network traffic at large time scale by using dual-related method." IEEE, Aug. 2016, pp. 1336–1340. [Online]. Available: http://ieeexplore.ieee.org/document/7603372/

[25] R. Babiarz and J.-S. Bedo, "Internet Traffic Mid-term Forecasting: A Pragmatic Approach Using Statistical Analysis Tools," in *NETWORKING 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, F. Boavida, T. Plagemann, B. Stiller, C. Westphal, and E. Monteiro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, vol. 3976, pp. 110–122. [Online]. Available: http://link.springer.com/10.1007/11753810_10

[26] J. Jiang and S. Papavassiliou, "Detecting Network Attacks in the Internet via Statistical Network Traffic Normality Prediction," *Journal of Network and Systems Management*, vol. 12, no. 1, pp. 51–72, Mar. 2004. [Online]. Available: http://link.springer.com/10.1023/B:JONS.0000015698.32353.61

[27] N. Wang, K. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, pp. 36–56, 2008. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4483669

[28] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "COPE: traffic engineering in dynamic networks," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, p. 99, Aug. 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1151659.1159926

[29] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," vol. 3. IEEE, 2001, pp. 1300–1309. [Online]. Available: http://ieeexplore.ieee.org/document/916625/

[30] D. Awduche, "MPLS and traffic engineering in IP networks," *IEEE Communications Magazine*, vol. 37, no. 12, pp. 42–47, Dec. 1999. [Online]. Available: http://ieeexplore.ieee.org/document/809383/

[31] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: responsive yet stable traffic engineering." ACM Press, 2005, p. 253. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1080091.1080122

[32] T. Benson, A. Akella, and D. A. Maltz, "Unraveling the complexity of network management." in *NSDI*, 2009, pp. 335–348.

[33] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015. [Online]. Available: http://ieeexplore.ieee.org/document/6994333/

[34] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined internet architecture: Decoupling architecture from infrastructure," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XI. New York, NY, USA: ACM, 2012, pp. 43–48. [Online]. Available: http://doi.acm.org/10.1145/2390231.2390239

[35] V. Kotronis, A. Gamperli, and X. Dimitropoulos, "Routing centralization across domains via sdn: A model and emulation framework for bgp evolution," *Computer Networks*, 2015.

[36] E. Lakiotakis, C. Liaskos, and X. Dimitropoulos, "Application-network collaboration using sdn for ultra-low delay teleorchestras," in *IEEE ISCC PEDISWESA Workshop 2017*, To Appear.

[37] P. Sermpezis and X. Dimitropoulos, "Inter-domain sdn: Analysing the effects of routing centralization on bgp convergence time," in *MAMA workshop (ACM SIGMETRICS)*, 2016.

[38] V. Kotronis, X. Dimitropoulos, R. Kloti, B. Ager, P. Georgopoulos, and S. Schmid, "Control exchange points: Providing qos-enabled end-to-end services via sdn-based inter-domain routing orchestration," in *Open Networking Summit (ONS)*, 2014.

[39] V. Kotronis, X. Dimitropoulos, and B. Ager, "Outsourcing the routing control logic: better internet routing based on sdn principles," in *ACM Hot Topics in Networks (HotNets)*, 2012.

[40] David H. Deans, "Why SDN revenue will grow by 72% CAGR through to 2020," Nov. 2016. [Online]. Available: https://www.telecomstechnews.com/news/2016/nov/15/sdn-revenue-will-grow-by-72-cagr-through-2020/

[41] "Seasonality," https://onlinecourses.science.psu.edu/stat510/node/67.

[42] H. A. Malki, N. B. Karayiannis, and M. Balasubramanian, "Short-term electric power load forecasting using feedforward neural networks," *Expert Systems*, vol. 21, no. 3, pp. 157–167, Jul. 2004. [Online]. Available: http://doi.wiley.com/10.1111/j.1468-0394.2004.00272.x

[43] J. W. Taylor, L. M. de Menezes, and P. E. McSharry, "A comparison of univariate methods for forecasting electricity demand up to a day ahead," *International Journal of Forecasting*, vol. 22, no. 1, pp. 1–16, Jan. 2006. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0169207005000907

[44] H. Hippert, C. Pedreira, and R. Souza, "Neural networks for short-term load forecasting: a review and evaluation," *IEEE Transactions on Power Systems*, vol. 16, no. 1, pp. 44–55, Feb. 2001. [Online]. Available: http://ieeexplore.ieee.org/document/910780/

[45] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, Oct. 2006. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0169207006000239

[46] X. Ding, Canu, S., and Denoeux, T., "Neural network based models for forecasting." in *In Proc. of Applied Decision Technologies Conf.*, 1995.

[47] A. Varga *et al.*, "The omnet++ discrete event simulation system," in *Proceedings of the European simulation multiconference (ESM'2001)*, vol. 9, no. S 185. sn, 2001, p. 65.

[48] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ser. Simutools '08. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 60:1–60:10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1416222.1416290

[49] "INET Framework." [Online]. Available: https://inet.omnetpp.org/

[50] D. Klein and M. Jarschel, "An openflow extension for the omnet++ inet framework," in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, pp. 322–329.

[51] "Open Networking Foundation. OpenFlow Switch Specification - Version 1.2." [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf

[52] S. S. Skiena, *The Algorithm Design Manual*, 2nd ed. Springer Publishing Company, Incorporated, 2008.

[53] L. Quan and J. Heidemann, "On the characteristics and reasons of long-lived internet flows." ACM Press, 2010, p. 444. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1879141.1879198

[54] N. Brownlee and K. Claffy, "Understanding Internet traffic streams: dragonflies and tortoises," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 110–117, Oct. 2002. [Online]. Available: http://ieeexplore.ieee.org/document/1039865/

[55] K. Thompson, G. Miller, and R. Wilder, "Wide-area Internet traffic patterns and characteristics," *IEEE Network*, vol. 11, no. 6, pp. 10–23, Dec. 1997. [Online]. Available: http://ieeexplore.ieee.org/document/642356/

[56] K.-c. Lan and J. Heidemann, "A measurement study of correlations of Internet flow characteristics," *Computer Networks*, vol. 50, no. 1, pp. 46–62, Jan. 2006. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1389128605001118

[57] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices." ACM Press, 2008, pp. 1–12. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1544012.1544023

[58] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures." ACM Press, 2004, p. 512. [Online]. Available: http://portal.acm.org/citation.cfm?doid=988672.988742

[59] A. W. Moore and K. Papagiannaki, "Toward the Accurate Identification of Network Applications," in *Passive and Active Network Measurement*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and C. Dovrolis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3431, pp. 41–54. [Online]. Available: http://link.springer.com/10.1007/978-3-540-31966-5_4

[60] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy, "Transport layer identification of P2p traffic." ACM Press, 2004, p. 121. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1028788.1028804

[61] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: automated construction of application signatures." ACM Press, 2005, p. 197. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1080173.1080183

[62] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: multilevel traffic classification in the dark," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, p. 229, Oct. 2005. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1090191.1080119

[63] T. Choi, C. Kim, S. Yoon, J. Park, B. Lee, H. Kim, H. Chung, and T. Jeong, "Content-aware Internet application traffic measurement and analysis." IEEE, 2004, pp. 511–524. [Online]. Available: http://ieeexplore.ieee.org/document/1317737/

[64] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese, "Network monitoring using traffic dispersion graphs (tdgs)." ACM Press, 2007, p. 315. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1298306.1298349

[65] Z. Li, R. Yuan, and X. Guan, "Accurate Classification of the Internet Traffic Based on the SVM Method." IEEE, Jun. 2007, pp. 1373–1378. [Online]. Available: http://ieeexplore.ieee.org/document/4288902/

[66] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 5, p. 5, Oct. 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1163593.1163596

[67] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning." IEEE, 2005, pp. 250–257. [Online]. Available: http://ieeexplore.ieee.org/document/1550864/

[68] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian Neural Networks for Internet Traffic Classification," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 223–239, Jan. 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4049810/

[69] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification." ACM Press, 2004, p. 135. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1028788.1028805

[70] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification." ACM Press, 2006, p. 1. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1368436.1368445

[71] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow Clustering Using Machine Learning Techniques," in *Passive and Active Network Measurement*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, O. Nierstrasz, C. Pandu Rangan, B. Steffen, D. Terzopoulos, D. Tygar, M. Y. Vardi, C. Barakat, and I. Pratt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, vol. 3015, pp. 205–214. [Online]. Available: http://link.springer.com/10.1007/978-3-540-24668-8_21

[72] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and discriminating between web and peer-to-peer traffic in the network core." ACM Press, 2007, p. 883. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1242572.1242692

[73] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, p. 50, Jun. 2005. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1071690.1064220

[74] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Offline/realtime traffic classification using semi-supervised learning," *Performance Evaluation*, vol. 64, no. 9-12, pp. 1194–1213, Oct. 2007. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0166531607000648

[75] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms." ACM Press, 2006, pp. 281–286. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1162678.1162679

[76] J. Postel, "User Datagram Protocol (RFC 768)," IETF Request For Comments, Aug. 1980.

[77] D. C. Plummer, "RFC 826: Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware," Nov. 1982. [Online]. Available: ftp://ftp.internic.net/rfc/rfc826.txt

[78] J. Postel, "RFC 793: Transmission Control Protocol," Sep. 1981.

[79] "Seasonality," http://statsmodels.sourceforge.net/.

[80] "Internet traffic Time Series Datasets," http://www3.dsi.uminho.pt/pcortez/series/.