UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Towards cognitive in-operation network planning

Fernando Morales Alcaide

Universitat Politècnica de Catalunya

Optical Communications Group

Towards Cognitive In-Operation Network Planning

Fernando Morales

Advisors:

Dr. Luis Velasco Esteban

Dr. Marc Ruiz Ramírez

A thesis presented in partial fulfilment of the requirements for the degree of

Philosophy Doctor

May 2nd, 2018

Optical Communications Group (GCO)

Universitat Politècnica de Catalunya (UPC)

C/ Jordi Girona, 1-3

Campus Nord, D6-107

08034 Barcelona, Spain

Agraïments

Primerament, voldria dedicar aquesta tesi a la meva família. El seu sacrifici incondicional és la raó per la qual hagi pogut arribar fins aquí. Gràcies de tot cor, n'estic orgullòs de vosaltres.

En segon lloc, voldria destacar la importància d'aquest període en el sí del Grup de Comunicacions Òptiques. Haver dut a terme la meva tesi en aquest grup m'ha permès compartir el dia a dia amb grans investigadors que han posat al meu abast tots els seus coneixement i eines en un ambient professional i acollidor. Gràcies.

Per últim, agraïr especialment als meus directors Luis Velasco i Marc Ruiz per la seva direcció impecable, pel seu suport continu i per totes les idees i consells que han compartit amb mi durant aquest temps.

Abstract

Next-generation internet services such as live TV and video on demand require high bandwidth and ultra-low latency. The ever-increasing volume, dynamicity and stringent requirements of these services' demands are generating new challenges to nowadays telecom networks. To decrease expenses, service-layer content providers are delivering their content near the end users, thus allowing low latency and tailored content delivery. Consequently, unseen metro and even core traffic dynamicity is arising with changes in the volume and direction of the traffic along the day.

A tremendous effort to efficiently manage networks is currently ongoing towards the realisation of 5G networks. This translates in looking for network architectures supporting dynamic resource allocation, fulfilling stringent service requirements and minimising the total cost of ownership (TCO). In this regard, *in-operation network planning* was recently proven to successfully support various network reconfiguration use cases in prospective scenarios. Nevertheless, additional research to extend in-operation planning capabilities from typical reactive optimization schemes to proactive and predictive schemes based on the analysis of network monitoring data is required.

A hot topic gaining attention is *cognitive networking*, where an elevated knowledge about the network could be obtained as a result of introducing data analytics in the telecom operator's infrastructure. By using predictive knowledge about the network traffic, in-operation network planning mechanisms could be enhanced to efficiently adapt the network by means of future traffic prediction, thus achieving *cognitive in-operation network planning*.

In this thesis, we focus on studying mechanisms to enable cognitive in-operation network planning in core networks. In particular, we focus on dynamically reconfiguring virtual network topologies (VNT) at the MPLS layer, covering a number of detailed objectives. First, we start studying mechanisms to allow network traffic flow modelling, from monitoring and data transformation to the estimation of a predictive traffic model based on this data. By means of these

traffic models, then we tackle a cognitive approach to periodically adapt the core VNT to current and future traffic, using predicted traffic matrices based on origin-destination (OD) predictive models. This optimization approach, named VENTURE, is efficiently solved using dedicated heuristic algorithms and evaluated under dynamic traffic network scenarios. Next, we extend VENTURE to consider core flows dynamicity as a result of metro flows re-routing, which represents a meaningful dynamic traffic scenario. This extension entails enhancements to coordinate metro and core network controllers with the aim of allowing fast adaption of core OD traffic models. Finally, we propose the network architectures needed to bring the previous mechanisms to experimental environments, using state-of-the-art network protocols such as OpenFlow and IPFIX.

The methodology employed to evaluate the previous work consists of a first numerical evaluation using a network simulator that was entirely devised and developed during this thesis. After the simulation validation, the experimental feasibility of the proposed network architectures is then assessed in a distributed test-bed.

It shall be mentioned that part of the work reported in this thesis has been done within the framework of European and National projects, namely METRO-HAUL (g.a. no. 761727, 2017-2020) funded by the European Commission, and SYNERGY (TEC2014-59995-R) funded by the Spanish Education and Science Ministry.

Resum

Els serveis d'internet de nova generació tals com la televisió en viu o el video sota demanda requereixen d'un gran ample de banda i d'ultra-baixa latència. L'increment continu del volum, dinamicitat i requeriments d'aquests serveis està generant nous reptes pels teleoperadors de xarxa. Per reduïr costs, els proveïdors de contigut estan disposant aquests més a prop dels usuaris finals, aconseguint així una entrega de contingut feta a mida. Conseqüentment, estem presenciant una dinamicitat mai vista en el tràfic de xarxes de metro amb canvis en la direcció i el volum del tràfic al llarg del dia.

Actualment, s'està duent a terme un gran esforç cap a la realització de xarxes 5G. Aquest esforç es tradueix en cercar noves arquitectures de xarxa que suportin l'assignació dinàmica de recursos, complint requeriments de servei estrictes i minimitzant el cost total de la propietat. En aquest sentit, recentment s'ha demostrat com l'aplicació de *in-operation network planning* permet exitosament suportar diversos casos d'ús de reconfiguració de xarxa en escenaris prospectius. No obstant, és necessari dur a terme més recerca per tal d'extendre in-operation network planning des d'un esquema reactiu d'optimització cap a un nou esquema proactiu basat en l'analítica de dades provinents del monitoritzat de la xarxa.

El concepte de *xarxes cognitives* es també troba al centre d'atenció, on un elevat coneixement de la xarxa s'obtindria com a resultat d'introduir analítica de dades en l'infrastructura del teleoperador. Mitjançant un coneixement predictiu sobre el tràfic de xarxa, els mecanismes de in-operation network planning es podrien millorar per adaptar la xarxa eficientment basant-se en predicció de tràfic, assolint així el que anomenem com a *cognitive in-operation network planning*.

En aquesta tesi ens centrem en l'estudi de mecanismes que permetin establir el cognitive in-operation network planning en xarxes de core. En particular, ens centrem en reconfigurar dinàmicament topologies de xarxa virtual (VNT) a la capa MPLS, cobrint una sèrie d'objectius detallats. Primer comencem estudiant mecanismes pel modelat de fluxos de tràfic de xarxa, des del seu monitoritzat i transformació fins a l'estimació de models predictius de tràfic. Posteriorment, i

mitjançant aquests models predictius, tractem un esquema cognitiu per adaptar periòdicament la VNT utilitzant matrius de tràfic basades en predicció de parells origen-destí (OD). Aquesta optimització, anomenada VENTURE, és resolta eficientment fent servir heurístiques dedicades i és posteriorment evaluada sota escenaris de tràfic de xarxa dinàmics. A continuació, extenem VENTURE considerant la dinamicitat dels fluxes de tràfic de xarxes de metro, el qual representa un escenari rellevant de dinamicitat de tràfic. Aquesta extensió involucra millores per coordinar els operadors de metro i core amb l'objectiu d'aconseguir una ràpida adaptació de models de tràfic OD. Finalment, proposem dues arquitectures de xarxa necessàries per aplicar els mecanismes anteriors en entorns experimentals, emprant protocols estat-de-l'art com són OpenFlow i IPFIX.

La metodologia emprada per evaluar el treball anterior consisteix en una primera evaluació numèrica fent servir un simulador de xarxes íntegrament dissenyat i desenvolupat per a aquesta tesi. Després d'aquesta validació basada en simulació, la factibilitat experimental de les arquitectures de xarxa proposades és evaluada en un entorn de proves distribuït.

Cal dir que part del treball mostrat en aquesta tesi ha estat realitzat en el sí de projectes de recerca europeus i nacionals, concretament METRO-HAUL (g.a. no. 761727, 2017-2020) finançat per la Comisió Europea, i SYNERGY (TEC2014-59995-R) finançat pel Ministeri d'Educació i Ciència.

Table of Contents

List of Figures

List of Tables

Page

Chapter 1

Introduction

1.1 Motivation

A new generation of internet services requiring stringent requirements, such as video on demand and live video streaming [Ru16], is changing the Internet traffic. The bandwidth demanded by these services is continuously growing: Cisco forecasts yearly global growth of 24% in the IP traffic between 2016 and 2021 [Cisco17]; by the end of that period, video traffic will account for 82% of the global IP traffic. The stringent requirement that these services demand is making content providers to carry traffic closer to end users by delivering it from metro networks. Metro-delivered traffic is growing faster than core-delivered traffic and will account for 35% of total end-user traffic by 2021.

An important part of this growth is caused by mobile traffic, experiencing a yearly global growth of 46%, twice as fast as fixed IP traffic [Cisco17]. In that regard, a tremendous effort is being carried out toward 5G networks aiming at facing a host of new challenges covering not only radio but also the non-radio segments of the network (metro and core). The number of nodes, the heterogeneity of the access technologies, the conflicting management objectives, the resource usage minimisation, and the division between limited physical resources and elastic virtual resources is driving a complete change in the methodology for efficient network management [5G].

From the metro-core point of view, the accommodation of these services is introducing an unprecedented dynamicity in the traffic involving changes not only in its volume but also in its direction over time. On statically, and even on reactively managed networks such dynamicity entails large overprovisioning, leading to large capital expenditures (CAPEX). With this in mind, operators are seeking for architectures that allow dynamic resource allocation, while fulfilling

services' requirements and minimising the total cost of ownership (TCO). Notwithstanding dynamic network operation might bring cost savings, dynamicity itself might cause network resources not to be optimally used. To solve that, *in-operation network planning* [Ve14.1] was proposed to make network resources available by re-configuring and re-optimising the network on-demand. Examples include re-optimisation [Gi15] and spectrum defragmentation in the optical layer [Gi14].

By looking at the MPLS layer, static virtual network topologies (VNT) have been commonly designed to cope with the off-line traffic forecast. VNTs consist of large packet-switching nodes (e.g., MPLS routers) connected through virtual links (*vlinks*) supported by static connections in the optical layer. In this context, in-operation planning algorithms for VNT reconfiguration such as threshold-based VNT reconfiguration [Ag15] have been proposed to reroute MPLS paths reactively by monitoring vlink capacity utilisation according to some configured threshold (e.g., 90%) and triggering a VNT reconfiguration once such threshold is exceeded. Although this reactive, threshold-based reconfiguration introduces OPEX savings when compared to a static VNT, transponder optimization is limited as a result of its local scope both topologically (per vlink adaptation) and in time (analyzing the current traffic). To achieve a superior VNT reconfiguration reactive approaches need to evolve to proactive ones capable of anticipating future traffic, thus allowing an efficient adaptation of the whole VNT to near-future traffic matrices.

Data analytics presents itself as an interesting option to achieve this. Data analytics is a field devoted to inspecting, cleansing, transforming, and modelling large and heterogeneous amounts of data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making. Data analytics has multiple facets and approaches, encompassing diverse techniques under a variety of names applied in several fields. In networking, the data source comes from monitoring the network nodes at the different network layers (e.g., packet or optical nodes). Algorithms based on monitoring the data plane could be used to elevate the cognitive abilities of in-operation network planning, providing means to adapt the network not only reactively but also proactively based on history data hence achieving a superior network configuration [5G]. Therefore, *cognitive in-operation network planning* appears as a promising solution for the next generation of network management.

1.2 Goals of the PhD thesis

In light of the above, this PhD thesis focuses on exploring cognitive in-operation network planning by combining data analytics based on monitoring data with optimisation techniques to allow applying the *observe-analyse-act* loop (OAA) in the

core VNT, adapting it not only to current but also to future traffic conditions (Fig. 1-1).

We propose to monitor traffic flows in core networks to obtain monitoring data used for knowledge discovery purposes (*observe*). Next, centralised data analytics algorithms are executed the core network controller based on monitoring data aiming at extracting information about the monitored traffic in the form of origin-destination (OD) predictive traffic models (*analyse*).

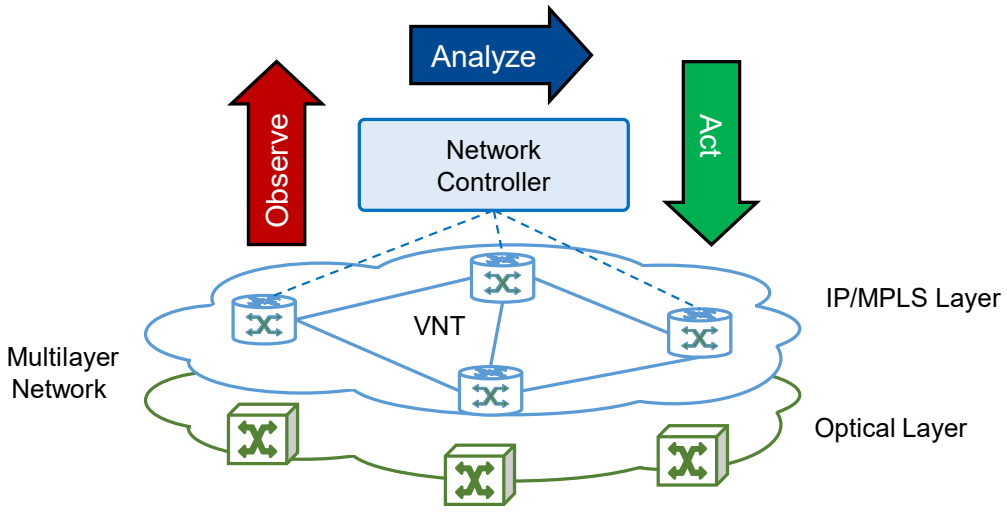

Fig. 1-1 Applying the OAA loop [Ve16] in multilayer optical networks.

To adapt the VNT to current and future traffic conditions while ensuring the target grade of service, the maximum predicted OD traffic matrix is computed based on the obtained predictive models and then used as input of a VNT reconfiguration problem based on traffic prediction (*act*).

To ensure the quality of core OD traffic models under changing traffic patterns, we propose to bring data analytics to the neighbouring metro areas to obtain metro-flow predictive models. These models, similarly computed in the metro controllers, are conveniently disseminated to the core controller and then aggregated to form updated OD core predictive models.

Three specific goals are defined to achieve this:

G1 – Traffic Modeling

This goal is divided into two sub-goals:

- **G1.1**: To study mechanisms for core and metro network traffic flow modelling.
- **G1.2**: To study mechanisms for efficiently rebuilding obsolete core network traffic flow models under changing traffic, aggregating several metro-flow traffic models from neighbouring metro network areas.

G2 – VNT Reconfiguration Based on Traffic Prediction

This goal is divided into three sub-goals:

- **G2.1:** To study how to periodically reconfigure the VNT to adapt it to current and predicted traffic matrices in a cost-saving manner (referred to as the VENTURE problem).
- **G2.2**: To study advanced solving techniques for the VENTURE problem.
- **G2.3**: To solve the VENTURE problem using the aggregated metro-flow traffic prediction under changing traffic scenarios.

G3 – Experimental validation

- This goal concentrates on studying and finding the best way to bring the algorithms from the previous goals to experimental environments.

A summary of the goals of the thesis is shown in Table 1.

Table 1: Thesis goals

	Goals	
G1 Traffic Modeling	**G1.1**	Traffic flow modelling.
	G1.2	Core-flow models based on metro-flow model aggregation.
G2 VNT Reconfiguration Based on Traffic Prediction	**G2.1**	To periodically adapt the VNT to the near future traffic (VENTURE problem).
	G2.2	Advanced solving techniques for VENTURE
	G2.3	Solve VENTURE based on aggregated metro-flow traffic prediction.
G3 Experimental validation	Bring the algorithms from previous goals to experimental environments	

It is worth mentioning that this thesis is encompassed in the research activity of the Optical Communications Group (GCO) at Universitat Politecnica de Catalunya. Specifically, it continues with the work of two previous PhD thesis recently done within the group. In the one hand, it continues the work of Dr Alberto Castro ([CaPhD]) devoted to the design of provisioning and re-optimization algorithms in elastic optical networks. On the other hand, it partially continues the work of Dr Lluís Gifre ([GiPhD]) devoted to experimentally assessing in-operation network planning architecture and algorithms.

1.3 Methodology

To carry out the studies needed to meet the goals of this thesis, the methodology illustrated in Fig. 1-2 will be followed. As the starting point of each study, an idea related to a thesis objective is conceived and formally stated. Due to the nature of this thesis, a data analytics algorithm and an optimisation problem are devised, being the optimisation problem formally modelled as a Mixed Integer Linear Programming problem (MILP). The data analytics algorithm is implemented in MATLAB or C++, whereas the mathematical model is solved using IBM's CPLEX solver [CPLEX]. Because the realistic scenarios tackled lead to large problem sizes in most of the studies, MILP formulations require long computation times prohibitive for in-operation planning purposes. Therefore, aiming at achieving practical heuristic algorithms are required. To that end, once a MILP formulation is validated, a heuristic algorithm is designed and implemented in C++.

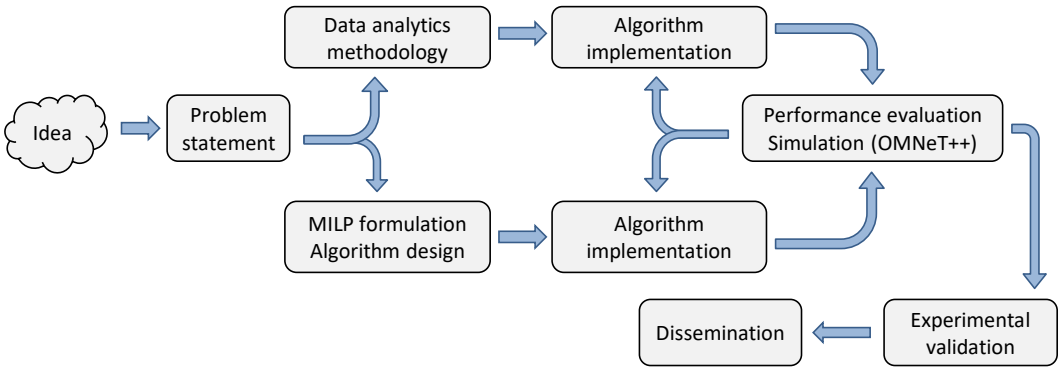

Fig. 1-2 Methodology used in this PhD thesis.

Data analytics and optimisation algorithms are considered jointly as our proposed solution to the problem. These algorithms are integrated into an event-driven network simulator based on OMNeT++ [OMNeT] that has been entirely designed and developed within this thesis. It allows evaluating the performance of the solution and, if required, revising and improving the algorithms. Then, the performance of the solution is compared against a certain benchmark (e.g., other solutions based on state-of-the-art procedures). Once the solution is validated through simulation, it is brought to a real environment where it is experimentally assessed. Finally, relevant results are disseminated and considered as the conception of a new idea requiring further research.

1.4 Thesis outline

The remainder of this thesis is organized as follows:

Chapter 2 presents a brief introduction to multilayer optical transport networks, including optical transmission technology, network topologies, transponder devices and Elastic Optical Networks (EONs). Then, the multilayer network architecture is described followed by an overview of the evolution from off-line network operation to in-operation network planning. This chapter ends by presenting the mathematical optimisation and data analytics tools applied to solve the problems faced along this thesis.

Chapter 3 explores the state-of-the-art used as a reference point in the development of this thesis. First, the latest research on network traffic modelling is explored including those works addressing the aggregation of predictive models. The chapter concludes by presenting the current status of VNT reconfiguration mechanisms.

Chapter 4 presents the network simulation environment used to evaluate the algorithms proposed in this thesis. The simulator was devised and developed jointly with this thesis.

Chapter 5 focuses on network traffic flow modelling. We explore the best ways to monitor, aggregate and process monitoring data to obtain quality traffic predictive models.

Chapter 6 shows how predictive OD traffic models can be included in the process of core VNT reconfiguration to adapt it not only to current but also to future traffic conditions. We accomplish this by solving a VNT reconfiguration problem based on traffic prediction (VENTURE).

Chapter 7 is devoted to studying advanced solving techniques for the VENTURE problem. Metaheuristics are proposed and evaluated through simulation aiming at finding the most suitable way of adapting the VNT over time.

Chapter 8 presents the problem of rebuilding core OD traffic models once they become obsolete as a result of metro-flow traffic rerouting from adjacent metro areas. By correlating metro and core traffic, metro-flow models can be aggregated to obtain updated core OD traffic models while in-operation.

Chapter 9 focuses on devising and assessing network and control architectures to bring the algorithms proposed in the previous chapters to experimental environments.

Finally, Chapter 10 draws the main conclusions of the thesis.

Chapter 2

Background

To perceive how optical transport networks are managed, the multilayer network architecture is first introduced to define how optical resources can be shared among different services thus defining virtual network topologies (VNT). The definitions of network operation and planning are then presented, continuing by a brief overview of the network life-cycle going from classic off-line network operation to current in-operation network planning.

Next, we introduce the concepts and technologies considered for the development this thesis. First, optical transport networks are presented. Dense Wavelength Division Multiplexing (DWDM) transmission technologies and later Elastic Optical Networks (EON), which promise a more efficient usage of the optical resources are briefly introduced. Finally, the MPLS layer is also reviewed.

Finally, we present the mathematical background needed to understand the algorithms presented in this thesis. First, we briefly introduce the topic of mathematical network optimization, comprising mathematical programming and heuristics. Second, we review the probability and statistics theory playing a role in the data analytics methods that will be seen in the next chapters.

2.1 Network transport technology

2.1.1 Transmission technology

An optical network is a network composed of optical nodes connected using optical fibers. In these networks, the information is transmitted as an optical signal; in the network nodes, the signal can be switched optically or converted to the electrical domain.

The Dense Wavelength Division Multiplexing (DWDM) technology allows transmitting different data flows on different optical wavelengths. Most DWDM systems currently use the frequency region around 1550 nm, because this is one of the frequency regions where the signal attenuation reaches a local minimum. Fig. 2-1 shows an example of the DWDM technology.

DWDM systems with channel spacings ranging from 12.5 GHz to 100 GHz have been specified [G694]. With that technology, the number of optical wavelength channels being multiplexed onto a single fiber ranges between 50 and 400.

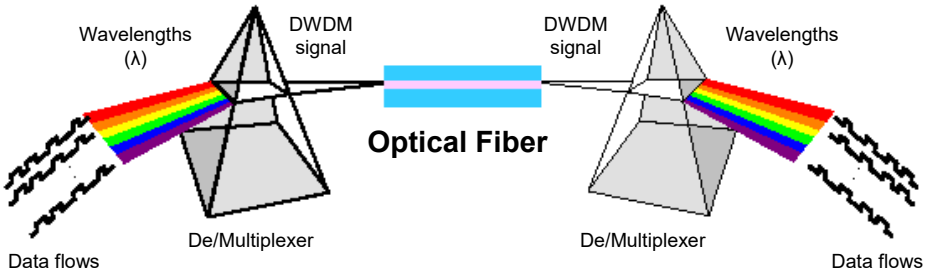

Fig. 2-1 DWDM Technology.

Light emitters (usually semiconductor lasers) are key components in an optical network. They convert an electrical signal into the corresponding light signal, on a single wavelength, that can be injected into the fiber. Besides, a DWDM system uses a multiplexer at the transmitter side to multiplex the different wavelengths together in a bundle, and a demultiplexer at the receiver side to split them apart. An optical fiber transmits optical signal through long distances. On a typical 80 channel DWDM system transporting 10Gbit/s per wavelength, the maximum distances that can be transmitted without regeneration are about 2000 km [Pe04].

In all-optical or *transparent networks,* optical connections are established between end-nodes assigning them a specific optical channel without any intermediate electronic processing.

When a connection request (or demand) between a source and a destination arrives, an optical connection is established in the optical network; these optical connections are commonly referred to as *lightpaths*.

2.1.2 Network topology

The first DWDM systems were point-to-point systems. The introduction of Reconfigurable Optical Add/Drop Multiplexer (ROADM) in the transport networks allows them to be configured in ring-based topologies. A ROADM allows dropping a specific wavelength out of the bundle of DWDM-multiplexed signals and adding

another channel on the same wavelength (Fig. 2-2a). A ROADM also allows for remote configuration and reconfiguration.

The introduction of sophisticated optical devices such as Wavelength Selective Switches (WSS) [Ts06] made possible to build evolved ROADM architectures and optical cross-connects (OXCs), the key element to build optical mesh-based networks (Fig. 2-2b) [Ro08].

Ring-based networks present lower capacity efficiency than mesh networks; mesh networking allows connections to be routed over shorter paths. In this regard, mesh-based networks have been extensively used in packet-based networks due to their high efficiency and flexibility.

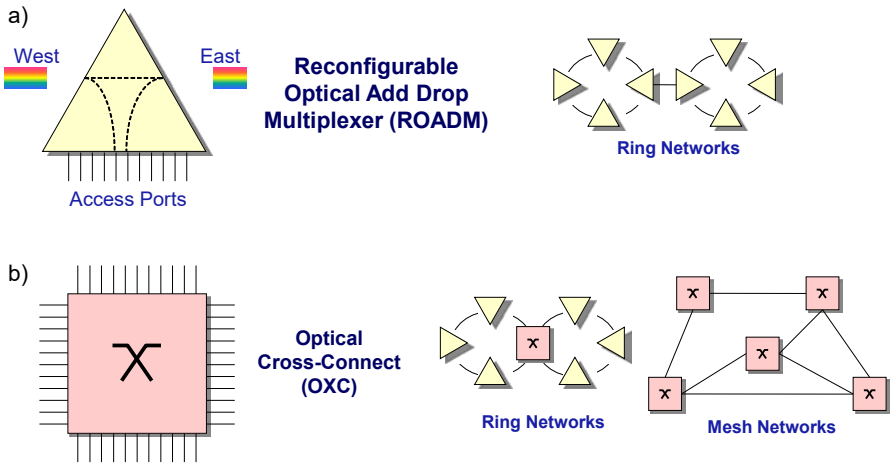

Fig. 2-2 Optical nodes and topologies.

2.1.3 Transponders

Transponders are optical devices responsible for transforming electrical flows into optical signals and vice-versa; therefore, they are used as end-points of lightpaths. Depending on their properties, different kinds of transponders are described in the literature. The selection of the kind of transponder is related to the characteristics of the lightpaths conveyed through the optical network [Ri12]. In optical networks where all lightpaths are of the same capacity, e.g. 100 Gb/s, Fixed Transponders (FT) can be used, and just one type of transponder needs to be kept in the inventory. However, when lightpaths have different capacities, e.g. 40/100/400 Gb/s, FTs of different capacities have to be installed in the network nodes, and different types of transponders have to be maintained in the inventory thus, significantly increasing network operational expenditures (OPEX).

Moreover, since the capacity of demands and transponders must coincide, compatible transponders need to be available in the end nodes of that connection.

Even though FTs could be available in those nodes, lightpaths could not be set up if they are not of the required capacity.

As an alternative solution, Bandwidth-Variable Transponders (BVT) enhances FTs' characteristics by enabling dynamic configuration of their capacity and can be used in a range of capacities, e.g. 40/100/400 Gb/s. In such way, the inventory contains transponders of a single kind thus, notably reducing network OPEX.

However, BVTs are underutilised when lightpaths require lower capacities than that of the BVTs. For instance, a 400 Gb/s BVT can end a lightpath of 400 Gb/s at the most; when lower capacities are requested, the residual capacity remains unused.

To increase the flexibility of BVTs and improve their utilisation, a new kind of transponder, named as Sliceable BVT (SBVT), has been proposed in the literature [Ji12], [Nap15]. SBVTs consist of a set of modules, each able to support an independent lightpath. SBVTs impose three constraints over the supported lightpaths: *i*) the total number of lightpaths cannot exceed the number of modules in the SBVT; *ii*) the total capacity of the supported lightpaths cannot exceed the total switching capacity of the SBVT; and *iii*) the spectrum allocated to each lightpath cannot overlap [Sam15]. For instance, an SBVT of 400 Gb/s with 4 internal modules can support four lightpaths of 100 Gb/s, or two lightpaths of 100 Gb/s plus a third one of 200 Gb/s, and so on.

Different modulation formats provide a series of trade-offs and drawbacks. For instance, the 16-QAM modulation format ($B_{16\text{-}QAM}$=4 b/s/Hz) needs half spectrum width than QPSK (B_{QPSK}=2 b/s/Hz) to transmit the same bit rate. Whereas 16-QAM has worse receiver sensitivity than QPSK thus, limiting the distance of optical connections.

2.1.4 Elastic Optical Networks

[G964] standardisation defines a flexible grid (previously introduced in [Li11]) to realise Elastic Optical Networks (EONs). EON requires from specific components such as Bandwidth-Variable (BV) Spectrum Selective Switches to build BV-OXCs. The optical spectrum is divided into frequency slices, which are portions of the spectrum with a fixed width (e.g., 6.25 GHz). The central frequency defines where the assigned spectrum is centred, and it allows positioning the slices allocated to a connection. A subset of contiguous (adjacent) frequency slices is called a *frequency slot,* and it is characterised by its centre frequency and the number of slices that contains. To illustrate the concepts introduced above, Fig. 2-3 represents the spectrum of an optical fiber link in EON.

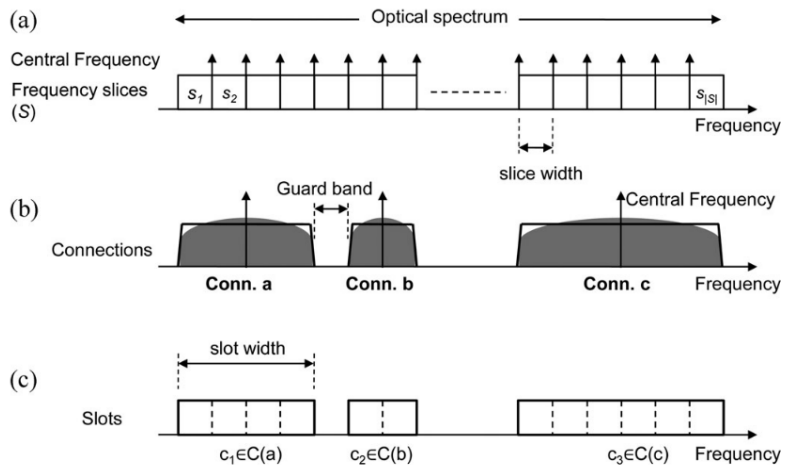

Fig. 2-3 Logical representation of a fiber link.

Thanks to this, an EON can adjust the slot allocation to varying traffic conditions over time, thereby creating a network scenario where slots are both switched and re-dimensioned according to temporary traffic requirements. Therefore, traffic demands are assigned a frequency slot according to their requested bit-rate and the selected modulation technique [Ji10].

Let us introduce some notation related to graph theory. Let graph $G(N, E)$ represent the topology of an EON, where N is the set of OXCs and E the set of optical links connecting two OXCs, and let S be the set of frequency slices available in each link $e \in E$. Then, given a connection request d requesting for $B(d)$ Gb/s, the spectral resources that need to be allocated are a function of the spectral efficiency (B_{mod} in b/s/Hz) of the chosen modulation format.

Without loss of generality, given B_{mod} and the spectrum granularity (e.g., 6.25 GHz), we compute the number of contiguous slots required by connection request d, denoted as $S(d)$, as in [Ji10]:

$$S(d) = \left\lceil \frac{B(d)}{B_{\text{mod}} \cdot 6.25} \right\rceil \tag{2-1}$$

It must be mentioned that eq. (2-1) tends to underestimate the number of slices required, as it assumes that $B(d)$ consists only of payload data. However, in general, this is not the case, as different overhead data (e.g., around 10% extra) may be required. Such overhead may vary according to the modulation format selected.

2.1.5 MPLS Networks

Multi-protocol Label Switching (MPLS) [RFC3031] is a type of data-carrying technique for high-performance telecommunications networks. MPLS directs data from one network node to the next based on path labels rather than long network addresses, avoiding complex lookups in a routing table. The labels identify virtual links (paths) between distant nodes rather than endpoints. MPLS can encapsulate packets of various network protocols, hence its name "multi-protocol". MPLS is a scalable, protocol-independent transport. In an MPLS network, data packets are assigned labels and packet-forwarding decisions are made solely on the contents of the label, without the need to examine the packet itself. This allows to create end-to-end circuits across any type of transport medium, using any protocol. The primary benefit is to eliminate dependence on a particular layer-2 technology.

An MPLS router that is located in the middle of an MPLS network and performs routing based only on the label is called a label switch router (LSR). An LSR is responsible for switching the labels used to route packets: when an LSR receives a packet, it uses the label included in the packet header as an index to determine the next hop on the label-switched path (LSP) and a corresponding label for the packet from a lookup table. The old label is then removed from the header and replaced with the new label before the packet is routed forward to the next MPLS router.

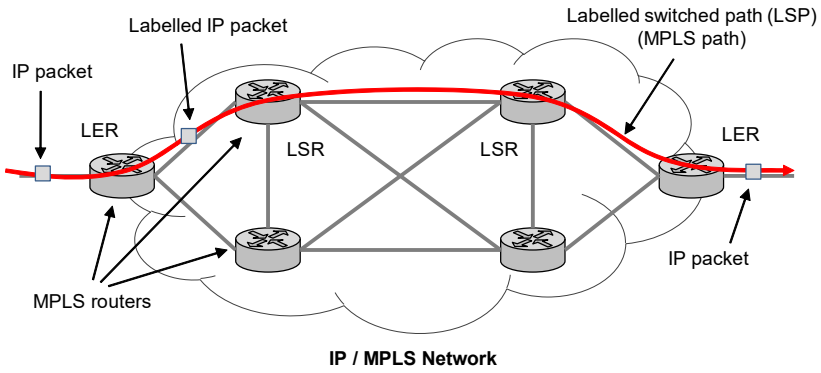

Fig. 2-4 Example of MPLS routing.

A label edge router (LER, also known as edge LSR) is a router that operates at the edge of an MPLS network and acts as the entry and exit points for the network. An ingress LER pushes an MPLS label onto incoming packets whereas an egress LER pop the MPLS label from outgoing packets; alternatively, this function may instead be performed by the last LSR directly connected to the LER. When forwarding an IP packet into an MPLS network, an LER uses routing information to determine the appropriate label to be pushed, labels the packet accordingly, and then forwards the labelled packet into the MPLS network. Likewise, upon receiving a labelled packet which is destined to exit the MPLS domain, the LER strips off the label and forwards the resulting IP packet using normal IP forwarding rules.

A label-switched path (LSP) is a path through an MPLS network, set up based on a particular forwarding equivalence class (FEC), which describes the set of packets which may be forwarded using the same MPLS label (e.g., those with the same destination IP address). The path begins at a label edge router (LER), which makes a decision on which label to prefix to a packet, based on the appropriate FEC. It then forwards the packet along to the next router in the path, which swaps the packet's outer label for another label, and forwards it to the next router. The last router in the path removes the label from the packet and forwards the packet based on the header of its next layer, for example IP. Due to the forwarding of packets through an LSP being opaque to higher network layers, an LSP is also sometimes referred to as an MPLS tunnel. Fig. 2-4 shows an example of an IP over MPLS (IP/MPLS) network routing a LSP.

Deploying MPLS networks provides an integrated approach for network traffic engineering (TE) allowing, among others. to determine the routes for traffic flows across a network based on the resources the traffic flow requires and the resources available in the network. This can be achieved by employing "constraint-based routing", where the LSP for a traffic flow is the shortest path that meets the resource requirements (constraints) of the traffic flow. In MPLS traffic engineering, the traffic flow has bandwidth requirements, media requirements, a priority versus other flows, and so on [Cisco99] [El01].

2.2 Network technologies and operation

2.2.1 Multilayer networks

The International Telecommunications Union Telecommunication Standardization Sector (ITU-T) defines in G.805 [G805] a reference layered transport network architecture with technology-independent relationships among functional entities. Therein, each network layer has a twofold role, namely, a server role to the client layer above it as well as a client role to the network layer below it. In brief, a subnetwork describes the capacity to associate a set of *connection points* (CPs) to convey the so-called characteristic information. With such an objective, two possible kinds of connection are defined: *i*) a *link connection* is a fixed and rigid connection between two CPs and *ii*) a *subnetwork connection* (SNC) is a flexible connection that may be set up and released by either the control or the management plane. As a result, a network connection is a concatenation of subnetwork and link connections delimited by a *termination connection point* (TCP) pair.

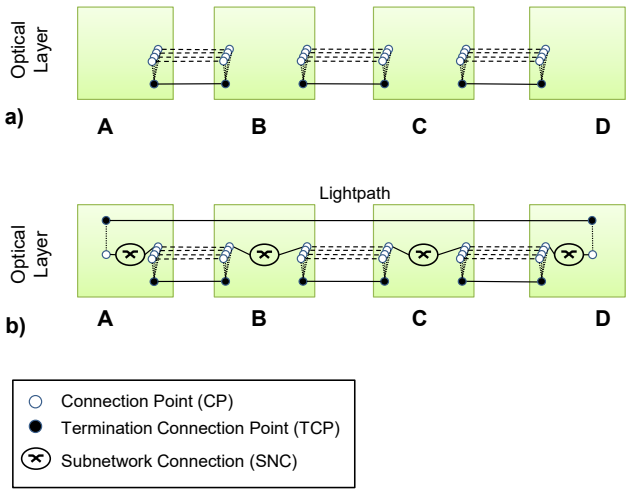

Fig. 2-5 Example of a single-layered network architecture.

A single-layered four-node optical network is exemplified in Fig. 2-5a. In such a scenario, link connections (representing the different frequency slot data links) associate CPs at remote neighbouring nodes; these link connection sets are bundled into network connections between remote TCPs. Let us suppose now that a lightpath is set up between ingress node A and egress node D (Fig. 2-5b). The incoming client signal at the optical node A is adapted and cross-connected using an SNC to an outgoing CP.

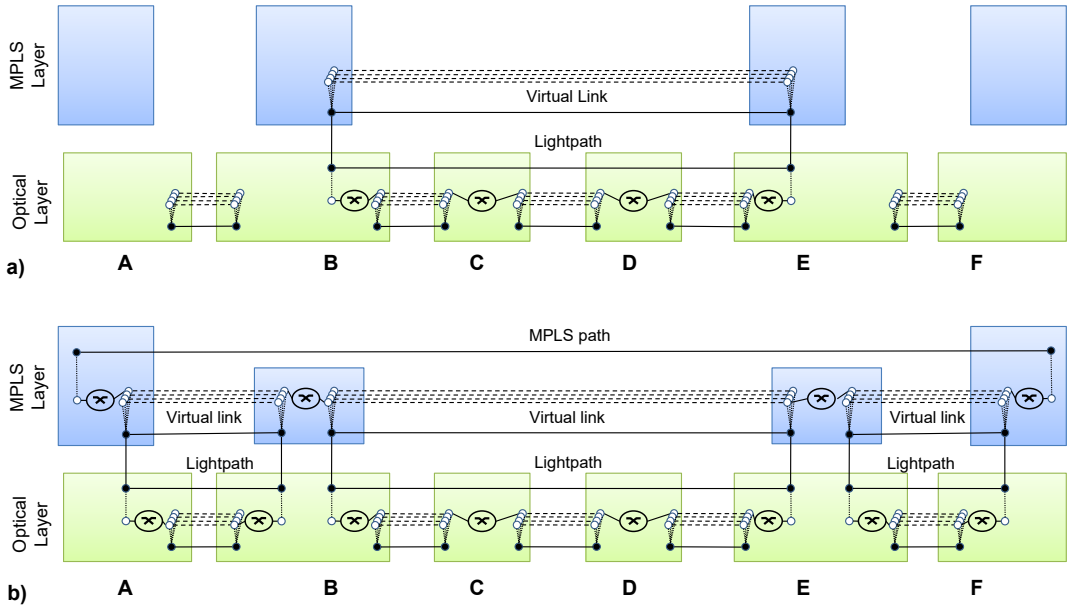

Fig. 2-6 Two-layer network architecture (a) before and (b) after setting up an end-to-end MPLS path supported by three vlinks.

This CP is, in turn, connected through a data link to an incoming CP in the neighbour of the node. At the intermediate nodes B and C, SNCs bind incoming and outgoing CPs, which should be mapped to the same frequency slot (assuming no spectrum conversion is performed). As soon as the signal reaches destination node D, it is cross-connected, adapted, and sent to the corresponding client access port.

Let us now present assume a two-layered network architecture with an optical server layer and a client MPLS aggregation layer on top. The MPLS layer allows the mapping of the client traffic to be transported over the optical layer. At the bottom, optical nodes provide client access ports, used to inject aggregated client flows to the network. The incoming electrical signal would be afterwards adapted, switched to and conveyed through the optical layer.

The architecture of the two-layered network under consideration is shown in Fig. 2-6, where an MPLS link connection commonly referred as a *virtual link* (vlink) is set up between MPLS routers B and E (Fig. 2-6a). Imagine that the requested bitrate becomes 1/4 of the total spectrum capacity. The incoming signal is adapted and further inserted into an outgoing lightpath, reaching in this way optical node E. Note that no processing is needed at intermediate MPLS routers C and D, as the signal optically bypasses them through the lightpath B–E. At the destination, the signal is demultiplexed, and the client signal is cross-connected, adapted, and delivered to the sink MPLS router. In the resulting scenario, routers B and E appear to be directly connected with an additional capacity of 3/4 of the total vlink capacity.

Further looking at Fig. 2-6b, the previously established vlink has now been used to create the transparent MPLS path between nodes A and F. To this end, two additional vlinks between nodes A–B and E–F are set up, providing the required connectivity at the MPLS layer. Specifically, an SNC associates incoming CPs with outgoing CPs at intermediate client nodes B and E. It is worth mentioning that an MPLS path supported on vlinks allocated in different frequency slots can be conveyed without spectrum conversion, as the signal is optical/electrically converted at the MPLS layer.

The previous example illustrates how the proposed architecture allows building MPLS network topologies, commonly referred to as virtual network topologies (VNT) where MPLS paths between end MPLS routers can be transparently routed. Figure Fig. 2-7 illustrates a high-level representation of the multilayer architecture, with a VNT on top of the optical network topology.

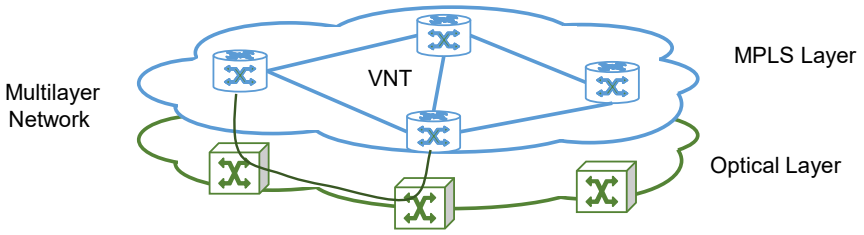

Fig. 2-7. Example of VNT supported by lightpaths on the optical layer.

2.2.2 Off-line planning

Planning (i.e., designing and dimensioning) the network consists in determining the nodes and links that need to be installed and which is the equipment to be purchased to serve the foreseen traffic while CAPEX. Because those tasks are done before the network or part of the network enters into operation, it is commonly known as *off-line planning*.

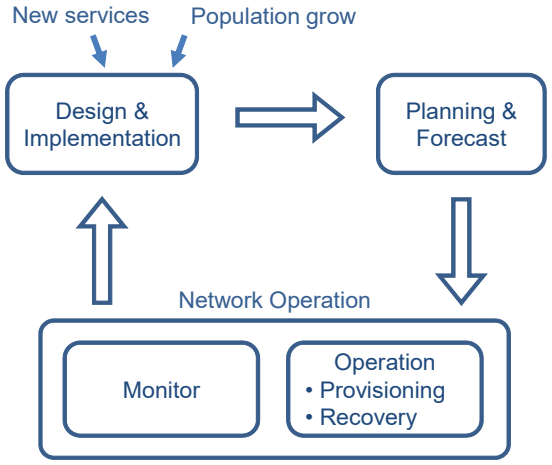

Fig. 2-8 Classical network life-cycle.

Indeed, the classical network life-cycle typically consists of several steps that are performed sequentially (Fig. 2-8). Starting with inputs from the service layer and the state of the resources in the already deployed network, a planning phase needs to be carried out to produce recommendations that the next phase uses to design the network for a given period. That period is not fixed, and the actual duration usually depends on many factors, which are operator and traffic type specific.

Once the planning phase produces recommendations, the next step is to design, verify, and manually implement (install) the network changes. While in operation, the network capacity is continuously monitored, and the obtained data are used as

input for the next planning cycle. In case of unexpected increases in demand or network changes, nonetheless, the planning process may be restarted.

2.2.3 Static vs dynamic network operation

Operation of the currently deployed transport networks is very complex since multiple manual configuration actions are needed for provisioning purposes (e.g., hundreds of thousands of node configurations per year in a mid-size network). In fact, transport networks are currently configured with big static fat pipes based on capacity over-provisioning, since they are needed for guaranteeing traffic demand and Quality of Service (QoS).

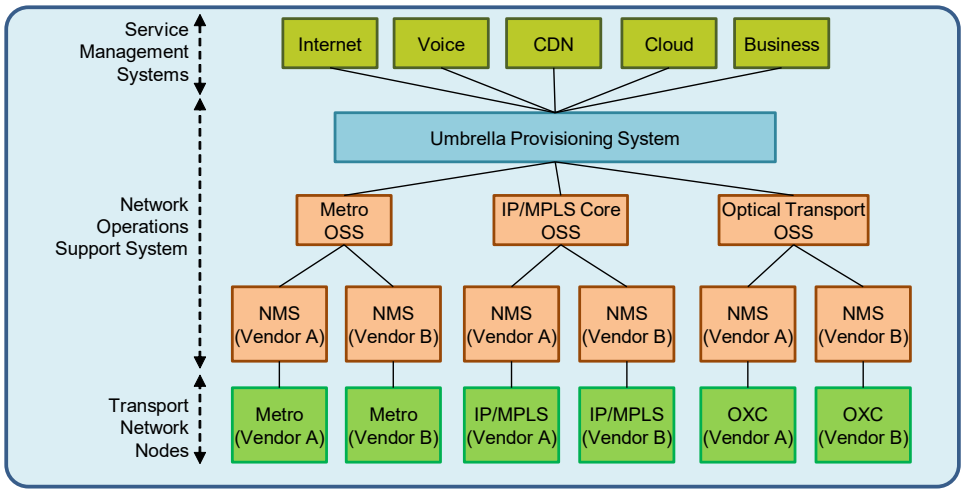

Fig. 2-9 Current static architecture.

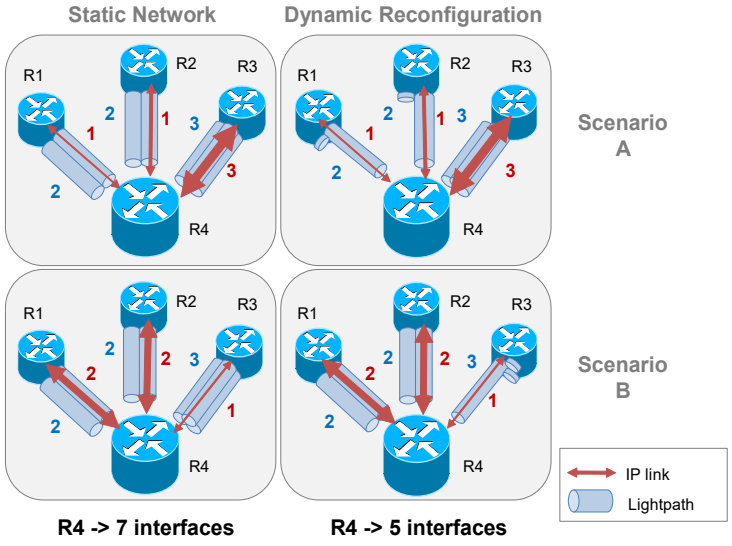

Fig. 2-10 An example of dynamic planning and reconfiguration.

Furthermore, network solutions from different vendors typically include a centralised service provisioning platform, using vendor-specific Network Management System (NMS) implementations along with an operator-tailored umbrella provisioning system, which may include a technology-specific Operations Support System (OSS) (Fig. 2-9). Such complicated architectures generate complex and long workflows for network provisioning: up to two weeks for customer service provisioning and more than six weeks for core routers connectivity services over the optical core network [Lo13].

Fig. 2-10 illustrates the fact that such static networks are designed to cope with the requirements of several scenarios, and predicted short-term increases in bandwidth usage thus, requiring capacity over-provisioning and significantly increasing CAPEX. It shows a simple network consisting in three routers connected to a central one through a set of lightpaths established on an optical network.

Two different scenarios are considered, although the same amount of IP traffic is conveyed in each of them. In the scenario A, router R3 needs three lightpaths to be established to transport its IP traffic toward R4, whereas R1 and R2 need only one lightpath each. In contrast to the scenario B, R1 and R2 need two lightpaths, while R3 needs only one lightpath.

In static networks, where lightpaths in the optical network are statically established, each pair of routers has to be equipped with the number of interfaces for the worst case, resulting in R4 being equipped with 7 interfaces. However, if the optical network can be dynamically reconfigured setting up and tearing down lightpaths on demand, each router can be dimensioned separately for the worst case, regardless of the peering routers. As a result, R4 would need to be equipped with only five interfaces, thus saving 28.5% of total interface costs. This example opens the opportunity to *dynamic network operation*.

2.2.4 In-Operation Network Planning

Traffic dynamicity entails setting up connections by allocating resources when requested and releasing them when the connections are torn down. As a result of repeating these operations, the available resources in the optical or the MPLS layer could not be optimally used, hence increasing the number of blocked connections due to the lack of resources and decreasing the grade of service.

Given the trade-offs of dynamically operating the network, the classical network life-cycle can be extended with *in-operation network planning* [Ve14-1] [GiPhd], as shown in Fig. 2-11. In contrast to off-line planning, *in-operation planning* relates to reconfiguration and optimization problems solved over a network which is in-operation. This new step takes care of network re-configuration and re-optimisation in real time during network operation, implementing the changes immediately in the network.

The main constraint of in-operation network planning problems is the stringent computation times in which solutions must be found, normally in the range of hundreds of milliseconds to minimise traffic disruptions. Additionally, note that in-operation planning algorithms are a subset of on-line algorithms; while on-line algorithms can access to arbitrary data, such as geographic locations or service duration periods, in-operation planning algorithms are constrained to access the data available in the control plane.

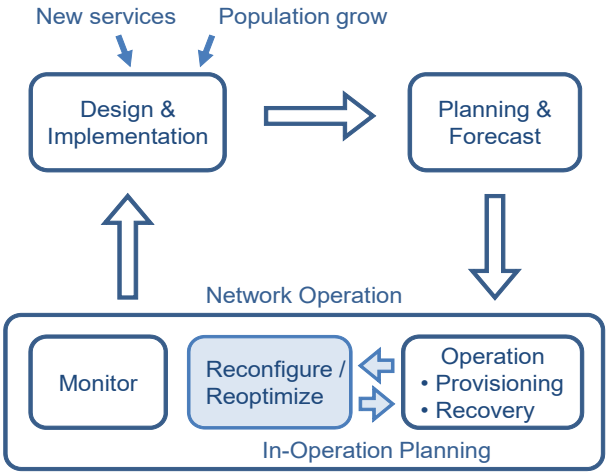

Fig. 2-11 Augmented network life-cycle.

2.2.5 Application-based network operations

Transport networks consist of different network segments (metro and core) and layers (IP/MPLS and optical), possibly managed by different vendors or using different technologies. To enable automatic provisioning and management of heterogeneous services over such multi- vendor/domain/layer networks, control and management planes need to be used. The control plane is in charge of automatically provisioning connections and reacting against failures in the network devices, while the management plane is responsible for providing fault, configuration, accounting/administration, performance and security management services to the network. Moreover, standard interfaces need to be available to avoid vendor specific solutions.

Nonetheless, to enhance network operation, a robust infrastructure capable of performing complex operations, such as the Application-based Network Operations (ABNO) architecture [RFC7491], standardized by the Internet Engineering Task Force (IETF), or a Software Defined Network (SDN) controller, as defined by the Open Networking Foundation (ONF) [ONF], is required. In this thesis, we assume the ABNO architecture since many network operators are adopting this solution.

The ABNO architecture consists of a number of standard elements with clearly defined functionality plus, a north-bound interface (NBI), a south-bound interface (SBI), and internal interfaces among modules (Fig. 2-12). This, enables distributing functional blocks across different processes for scalability purposes. The ABNO components need to work together to provide the network operations requested by the Network Management System (NMS) / Operations Support Systems (OSS) in charge of the network.

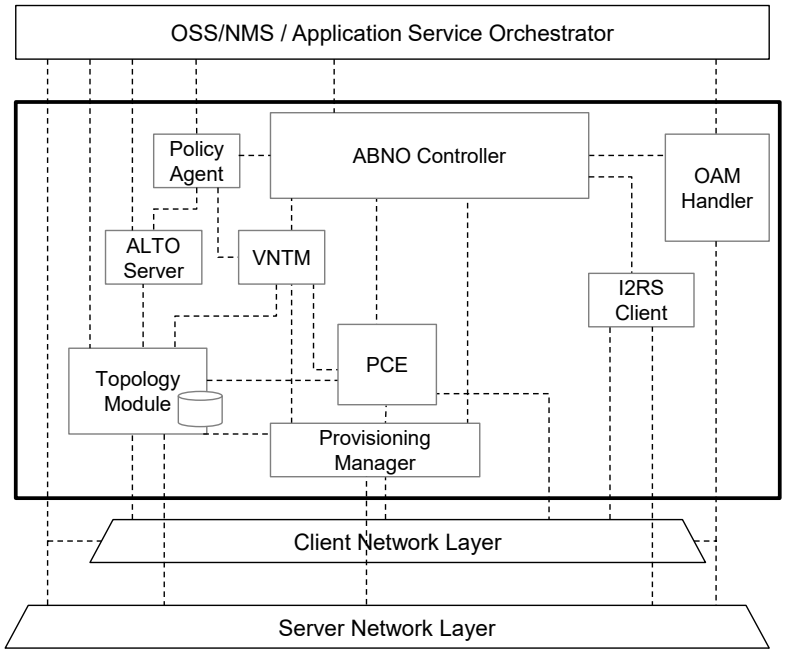

Fig. 2-12 ABNO architecture (reproduced from [RFC7491]).

A simplified ABNO architecture containing the elements required for the applications of in-operation planning developed along this thesis is shown in Fig. 2-13. Its elements are:

- The controller is the gateway that receives provisioning and service requests from the NMS / OSS through its NBI, translates them into simpler operations that the rest of functional components can deal with, and invokes those components according to implemented workflows to serve the requests.
- The policy agent is the entity that regulates the use and access to network functionalities and resources and checks the rights of the different elements of the ABNO architecture before they commit any operation. It is responsible for propagating these policies to other components in the ABNO architecture.
- The databases module stores the information related to the network. The TE Database (TED) contains the state of network resources. Besides, when

the Path Computation Element (PCE) is stateful (see PCE module below), the databases module contains an additional database, named as LSP Database (LSP-DB), maintaining information regarding current LSPs in the network; including the route, the bitrate, the spectrum allocation, the switching types, and the LSP constraints. The PCE updates the LSP-DB when connections are set-up, torn-down, or updated.

- The Path computation element (PCE) [RFC4655] is the entity devoted to perform constrained path computation on a graph representing the network by solving the RSA problem in the case of flexgrid-based networks for incoming connection requests as well as for solving optimization problems. It can be implemented with either stateless or stateful capabilities [draft-stateful]. In the stateless architecture, the PCE relies only on the TED to perform the path computation. The stateful PCE architecture extends the stateless one with the LSP-DB. Additionally, a stateful PCE may also include the active functionality that enables the PCE to modify the route and the frequency slot of already established LSPs.
- The VNT Manager (VNTM) [RFC5623] is in charge of creating and configuring VNTs in multi-layer networks by requesting to the PCE to set-up or tear down LSPs supporting the vlinks in such VNT [RFC5212].
- The Provisioning Manager (PM) is responsible for configuring the network devices to establish LSPs in the network**,** through the paths computed by the PCE. It uses ABNO's SBI to interact with a Generalized MPLS (GMPLS) [RFC3945] control plane or an SDN controller for directly programming the data path on each individual network node using protocols such as OpenFlow [OpenFlow].
- The Operations, Administration and Maintenance (OAM) Handler is responsible for receiving notifications from the network elements advertising for events such as failures, alarms or monitoring counters. When a message is received, the OAM Handler decides whether to invoke another component through the controller to deal with that event.

Some other elements are also included in the ABNO architecture. However, since they are not used in this thesis we refer the reader to [RFC7491] for further details. Additionally, a number of use cases are presented in [RFC7491] describing how ABNO can be applied to provide application-driven and NMS/OSS-driven network operations, such as multi-layer networking or global concurrent optimization.

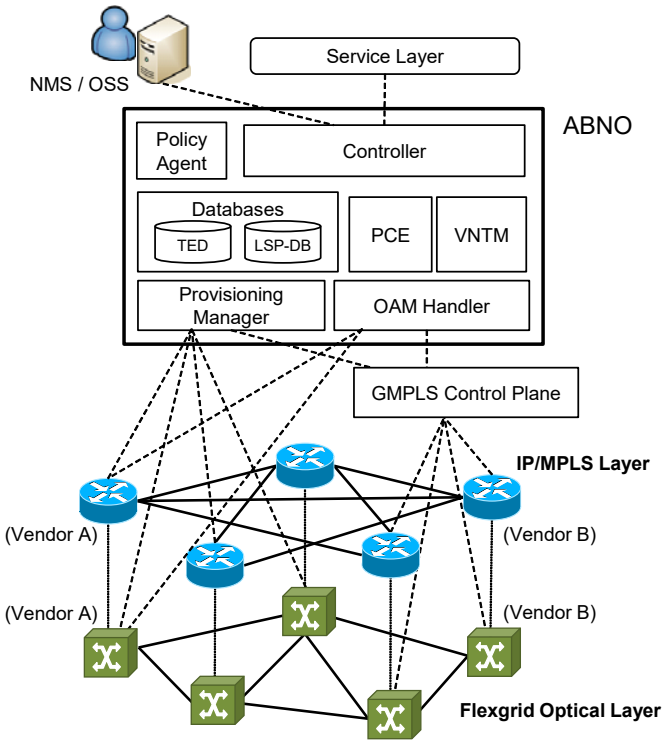

Fig. 2-13 ABNO-based Control and Management Plane Architecture

2.2.6 Network monitoring

Network monitoring consists of collecting different types of data from the different network layers with the aim of obtaining different performance metrics (e.g., to obtain bitrates measurements from the packet layer or to detect failures in the optical layer). In that regard, one of the first network monitoring protocols is the Simple Network Management Protocol (SNMP), developed by the IETF in 1990 [RFC1157]. Although it was initially designed to facilitate network device configurability, it proved to be a very popular network management protocol used for basic network monitoring (i.e., fault and performance management). As a consequence, a later review by the IETF about the SNMP protocol [RFC3535] revealed that operators were primarily using proprietary Command Line Interfaces (CLI) to configure network devices (as it had a number of practical advantages as opposed to SNMP). In addition, many equipment vendors did not provide the option to completely configure their devices via SNMP. Soon became clear that a more compact protocol than SNMP was needed to scale better for performance collection on IP networks. As a result of this, the NetFlow protocol was proposed by Cisco, today known as the IP Flow Information Export (IPFIX) protocol [RFC7011] which has become a standard implemented by many router and switch vendors.

Similar to the NetFlow Protocol, IPFIX considers a flow to be any number of packets observed in a specific timeslot and sharing a number of properties (e.g., same source, destination and protocol). Using IPFIX, devices like routers can inform a central monitoring station about their view of a potentially larger network. IPFIX is a push protocol, i.e. each sender will periodically send IPFIX messages to configured receivers without any interaction by the receiver. The actual data layout in IPFIX messages is widely configurable by the sender, based on special *monitoring templates* defining the monitoring data. In addition, the sender can also use custom user-defined data types in its messages, so the protocol is freely extensible and can adapt to different scenarios.

2.3 Network Optimization

In-operation network planning problems entail solving a series of complex decision-making problems, given the size of the networks and the wide margin of operation that current transmission technologies offer.

Because of this, it is crucial relying on mathematical methods not only capable of formally representing these problems, but also of providing high-quality solutions in practical times. In this section, we introduce two of these methods namely *mathematical programming* and *heuristics*, which will be applied in this thesis.

2.3.1 Mathematical programming

Mathematical Programming is the mathematical field that studies the selection of the best element x^* from a set of feasible solutions X, subject to some criteria or constraints [Ch83]. The quality of a choice $x \in X$ is relative to a function $f(x)$.

In particular, Linear Programming (LP) is a subset of Mathematical Programming where x is a real-valued vector, $f(x)$ is a linear function, and X is defined by a linear system of inequalities. More formally, a minimisation LP problem can be defined as follows:

$$z^* = \min f(x) \tag{2-2}$$

subject to:

$$Ax \leq b, x \geq 0 \tag{2-3}$$

where a feasible solution of the problem is any element x satisfying eq. (2-3). Thus, the optimal solution x^* of an LP problem is defined as follows:

$$x^* = \operatorname{argmin}\{f(x) : Ax \leq b, x \geq 0\} \tag{2-4}$$

Note that a problem can show one or multiple optimal solutions; it might even be the case where x^* does not exist. For instance, in a maximisation problem where feasible solutions can make $f(x)$ arbitrarily large, the problem is said to be *unbounded*. Contrariwise, when the set of feasible solutions is empty, the problem is said to be *infeasible*.

Integer Linear Programming (ILP) is the subfield of LP including integrality constraints. If the problem combines integer and real variables, it is named as *Mixed Integer Linear Programming* (MILP) problem. Finally, *Non-Linear Programming* (NLP) refers to the subfield of Mathematical Programming with non-linear function or constraints.

Regarding the complexity of these problems, a decision problem is said to be Non-deterministically Polynomial (NP) if any candidate solution can be verified in a polynomial number of steps relative to the input size by a deterministic Turing machine. There is an exponential upper bound on the number of steps needed to decide an NP-complete problem, whereas it is still an open issue whether this bound is polynomial (i.e., P=NP). Among NP, NP-complete subclass refers to the hardest decision problems, to which ILP was proven to belong [St82].

Several exact procedures have been developed to find the optimal solution of a Mathematical Programming problem. For example, the *Simplex algorithm* for LPs and *Branch and Bound* or *Branch and Cut* algorithms for ILPs [Ch83]. When a problem is largely-scaled (i.e., huge number of variables and constraints), it becomes impractical to solve it using the previously described algorithms. In that case, *decomposition methods* such as column and row generation and interior point methods can be used to solve them. In the case of NLPs, the use of interior point methods such as *Newton's barrier method* has been exploited for non-linear constrained problems. Although the output of these exact methods is the optimal solution, the required computation time tends to be too high for practical purposes when real-life instances need to be solved, even if commercial solvers are used such as CPLEX [CPLEX].

When the achievement of the optimal solution is not possible, some relaxation methods such as *Lagrangian relaxation* or *Randomized rounding* provide good-quality solutions by relaxing some integrality constraints, thus decreasing the problem's complexity. As an alternative to these relaxation methods, meta-heuristic methods have also been deeply studied to provide near-optimal feasible solutions [Bl03]. Next section provides the basic concepts of meta-heuristics, as well as the details of the meta-heuristics used in this thesis.

2.3.2 Heuristics

Heuristics is a technique designed for solving a problem more quickly when exact methods (i.e., mathematical programming) are too slow, or for finding an approximate solution when exact methods fail to find the optimal. This is achieved

by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut. Results about NP-hardness in theoretical computer science make heuristics the only viable option for a variety of complex optimisation problems that need to be routinely solved in real-world applications, such as ILPs (e.g., the *travelling salesman problem* for transport networks).

Although the optimality of the heuristic solutions cannot be guaranteed, the obtained solutions can be compared to those of exact methods obtaining the so-called *optimality gap*. In particular, the *relative* optimality gap is defined in eq. (2-5), where z^{best} is the cost of the best solution found by a particular heuristic procedure, typically expressed as a percentage. In the usual case that z^* is not available, a lower (upper) bound for the minimisation (maximisation) problem can be obtained and used instead (e.g. as in *branch-and-bound* exact solving method).

$$\frac{\left|z^* - z^{best}\right|}{\left|z^*\right|} \tag{2-5}$$

A *meta-heuristic* can be defined as an iterative master process that guides and modifies the operations of subordinate heuristics to produce high-quality solutions efficiently. Because of this, metaheuristics can be seen as optimisation frameworks useful for a wide variety of optimisation problem regardless of the particular context. This is achieved by encapsulating context-specific features in specific parts of the metaheuristic and leaving the master procedure to specify a generic search strategy in the *solution space* (i.e., the feasible region).

Among the multiple types of metaheuristics, neighbourhood-based metaheuristics (or single-solution metaheuristics) aim at exploring the solution space obtaining and improving a single solution at a time. This allows shorter computation times when compared to population-based metaheuristics (e.g., genetic algorithms), dealing with a set of solutions at the same time. Typically, a search strategy for a neighbourhood-based metaheuristic is defined in terms of *diversification* and *intensification*: the first aims at performing a wider exploration of the solution space, whereas the second aims at narrowing the search in the vicinity of known quality solutions.

Two subordinate procedures are mostly used in neighbourhood-based metaheuristics: the *constructive* and *local search* phases. Constructive algorithms generate solutions from scratch by adding components to an initially empty partial solution, until a solution is complete. Local search algorithms, instead, start with an initial solution and iteratively try to replace the current solution with a better solution in an appropriately defined neighbourhood of the current solution. Two different approaches can be followed to improve a solution: replace by the first improving solution found in the neighbourhood (*first-improvement*) or by the best solution found in the vicinity (*best-improvement*), usually offering a different trade-off between computational effort and solution quality.

Examples of neighbourhood-based meta-heuristics are *Simulated Annealing*, *Path Relinking*, *Tabu Search* [Gl03], the *Greedy Randomized Adaptive Search Procedure* (GRASP) [Fe02] and *Iterated Local Search* [Lo03], just to mention some.

Let us next present the two neighbourhood-based metaheuristics for combinatorial network optimisation that will be used in this thesis.

2.3.3 Greedy randomised adaptive search procedure

The Greedy Randomized Adaptive Search Procedure (GRASP) [Fe02] is a neighbourhood-based metaheuristic that builds multiple solutions from scratch (multi-start) and eventually returns the best one found. GRASP allows parameterising its search strategy by a parameter $\alpha \in [0,1]$ and encapsulates context-dependent cost computation in a greedy cost function (GCF). GRASP is selected due to its proven performance in solving combinatorial network optimisation problems [Pe13].

The procedure (detailed in Table 2-1) receives the GCF and α and starts initialising the best solution and its cost to default values (line 1). Next, a number of iterations are run until a given stopping criterion is met (lines 2-6). Stopping criteria can be either deterministic (e.g., a maximum number of non-improving iterations or running time) or stochastic, such as the probabilistic stopping criterion [Ri13]. At each iteration, a new local optimum is obtained by running first a constructive phase (line 3) followed by a local search procedure (line 4). The best solution found is updated (lines 5-6) and eventually returned (line 7).

Table 2-1: GRASP main algorithm

```
INPUT GCF, α
OUTPUT x^best
1:  x^best ← ∅
2:  while not stoppingCriterion() do
3:     x ← constructivePhase(GCF, α)
4:     x' ← localSearchPhase(x)
5:     if f(x') < f(x^best) then
6:        x^best ← x'
7:  return x^best
```

The constructive phase (shown in Table 2-2) details how a new solution is built from scratch according to the GCF and α. First, a new solution and a *candidate list* (CL) are initialised (lines 1-2). The CL contains a list of *candidates*, each one representing a *part* or *component* of a solution x. Candidates are sequentially added to an empty solution until it becomes complete. Contrarily to a first intuition, the final cost of selecting a candidate is not usually known *a priori*. This is because candidates' costs depend on the order in which they are selected: an early choice of an expensive candidate could lead to finding cheaper candidates in the future or vice-versa. In other words, candidates' costs are not independent or

uncoupling. Before this difficulty in knowing the real cost of adding a candidate u to a solution, a GCF is used to provide a greedy (local) cost estimation only taking into account previously selected candidates (i.e., GCF(u, x)).

After the *CL* is computed, all its elements are iteratively selected according to a greedy-randomised criterion (lines 3-7). At each iteration, the cost of each remaining candidate $u \in CL$ is recomputed according to GCF(u, x). Next, a *restricted candidate list* (RCL) is computed as a subset of CL according to eqs. (2-6)-(2-8) (line 4): a value of α close to 0 will define a greedy candidate selection, whereas a value close to 1 will lead to a randomised selection. One candidate is selected at random from the RCL and added to the partial solution (lines 5-7).

Table 2-2: GRASP constructive phase

```
INPUT GCF, α
OUTPUT x
1:  x ← ∅
2:  CL ← initializeCL()
3:  while CL ≠ ∅ do
4:     RCL ← computeRCL(GCF, α) (eq. (2-6)-(2-8))
5:     u ← selectAtRandom(RCL)
6:     x ← x ∪ {u}
7:     CL ← CL \ {u}
8:  return x
```

$$RCL(CL, \alpha) = \left\{ u \in CL : GCF(u, x) \leq GCF^{\min} + \alpha \cdot \left(GCF^{\max} - GCF^{\min} \right) \right\} \quad (2\text{-}6)$$

$$GCF^{\min} = \min_{u \in CL} GCF(u, x) \quad (2\text{-}7) \qquad GCF^{\min} = \min_{u \in CL} GCF(u, x) \quad (2\text{-}8)$$

2.3.4 Iterated local search

Iterated Local Search (ILS) is a neighbourhood-based metaheuristic that builds an initial solution from scratch and then iteratively perturbs it at random aiming at finding quality solutions [Lo03]. ILS allows parameterising the strength of the perturbations by a *strength* parameter inducing its search strategy: strong perturbations allow moving further in the solution space (diversification) whereas small ones allow narrowing down the search to a particular region (intensification). Although sophisticated versions of ILS include tunning the strength parameter at runtime, for the sake of simplicity we present a version with a constant strength parameter. This metaheuristic acts as a black box, only requiring three functions to run: a constructive, a perturbation and a local search procedure. The performance of ILS in solving combinatorial network optimisation problems is also present in the literature [Wu15].

Table 2-3: ILS main algorithm

```
INPUT strength
OUTPUT x^best
1:  x ← constructivePhase()
2:  x ← localSearchPhase(x)
3:  x^best ← x
4:  while not stoppingCriterion() do
5:     x^pert ← perturb(x, strength)
6:     x ← localSearchPhase(x^pert)
7:     if f(x) < f(x^best) then
8:        x^best ← x
9:  return x^best
```

The algorithm in Table 2-3 starts by finding an initial local optimum by first running a constructive phase followed by a local search (lines 1-3). After this, a number of iterations are run until some stopping criterion is met (lines 4-8). At each iteration, a randomised perturbation is applied according to the strength parameter thus obtaining a new feasible solution (line 5). It is worth noting that some versions of ILS allow reaching infeasible solutions after a perturbation is applied; this is done in the hope of moving further across the feasible region once an infeasible solution is repaired. Once a new solution is obtained, a local search procedure is applied to the current solution x to reach a local optimum (line 6). The best solution found is updated (lines 7-8) and eventually returned (line 9).

2.3.5 The RSA problem

Although the problems addressed in this thesis relate to MPLS-over-optical multilayer networks, we stress the resolution and analysis of these problems at the MPLS layer, assuming that well-known subproblem is solved in the optical layer every time that a lightpath is needed to upgrade the capacity of the core VNT. To facilitate the understanding of the presented algorithms in the MPLS layer, let us introduce the *routing and spectrum allocation problem* (RSA), solved every time that a lightpath is needed to support the VNT.

The Routing and Spectrum Allocation Problem (RSA) problem consists in finding a feasible route and spectrum allocation for one or multiple demands. Similar to the Routing and Wavelength Assignment (RWA) problem in DWDM networks, the spectrum continuity constraint must be enforced. In the case of EONs, the spectrum allocation is represented by a slot, and thus, in the absence of spectrum converters, the same slot must be used along the links of a given routing path; this is called as *continuity constraint*. Besides, the allocated spectrum slices must be contiguous; this is called as spectrum *contiguity constraint*. The RSA problem was proved to be NP-complete in [Chr11] and [Wa11]. As a consequence, research has been carried out to find efficient methods to solve real size RSA instances [Ve12], [Ve16].

Due to the spectrum contiguity constraint, RWA problem formulations developed for DWDM networks are not applicable for RSA in EONs, and they need to be adapted to include that constraint. Although several works can be found in the literature presenting ILP formulations for RSA, here we rely on those in [Ve12] since their approach, based on the assignment of slots, allows efficiently solving the RSA problem.

The definition of slot can be mathematically formulated as follows. Let us assume that a set of slots $C(d)$ is pre-defined for each demand d, which requests n_d slices. Let q_{cs} be a coincidence coefficient which is equal to 1 whenever slot $c \in C$ uses slice $s \in S$, and 0 otherwise. Hence, $\forall c \in C(d)$ the spectrum contiguity constraint is implicitly imposed by the proper definition of q_{cs}, stated as follows:

$$\forall i, j \in S, i < j, q_{ci} = q_{cj} = 1 \Rightarrow q_{ck} = 1, \forall k \in \{i,..., j\}, \sum_{s \in S} q_{cs} = n_d \quad (2\text{-}9)$$

In this thesis, we consider that each set $C(d)$ consists of all possible slots of the size requested by d that can be defined in S. Since $|C(n)|=|S|-(n-1)$, the size of the complete set of slots C that needs to be defined is $|C|=\sum_{n \in N}[|S|-n+1] < |N| \cdot |S|$.

The algorithm in Table 2-4 computes $C(d)$. Note that slot computation is trivial and thus, no additional complexity is added to the pre-computation phase.

Table 2-4 Pre-Computation of C(d)

```
INPUT S, d
OUTPUT C(d)
1:  Initialize: C(d) ← 0[|S|-nd + 1·|S|]
2:  for each i in [0, |S|-nd] do
3:     for each s in [i, i+nd-1] do
4:        C(d)[s]=1
5:  return C(d)
```

Therefore, we can define the RSA problem as the problem that finds a proper lightpath (i.e., a route and a slot) for each demand from a given set so that the number of active slices in the assigned slot guarantees that the bitrate requested by each demand can be transported. Note that by pre-computing the set of slots that can be assigned to each demand, the complexity added by the contiguity constraint is removed. Finally, without loss of generality, we can consider that guard bands are included as a part of the requested spectrum (i.e., in n_d).

A very basic RSA problem consists in finding a lightpath for every demand in a given traffic matrix with the objective of minimising or maximising some utility function. Several alternatives for this problem may exist, for instance, we can assume that all the traffic matrix needs to be served, or some demands can be blocked, i.e. not served. Note that additional characteristics, such as selecting the modulation format and limiting lightpath reach could be defined. The problem can be formally stated as follows.

Given:

- A connected graph $G(N, E)$, where N is the set of locations and E is the set of optical fibers connecting two locations,
- the characteristics of the optical spectrum (i.e., spectrum width and frequency slice width) and the set of modulation formats,
- a traffic matrix D with the amount of bitrate exchanged between each pair of locations in N.

Output: the route and spectrum allocation for each demand in D.

Objective: one or more among:

- Minimise the amount of bitrate blocked,
- Minimise the total amount of used slices.

In the following, we present an ILP model for the above problem, based on the formulations in [Ve12]. Note that since the topology is given, we can pre-compute a set of k distinct paths for each of the demands in the traffic matrix and hence, the formulation is usually known as link-path [Pi04]. Moreover, because of the use of pre-computed slots for each demand, we call this formulation as link-path slot-assignment (LP-SA).

The following sets and parameters have been defined.

Topology:

N Set of locations, index n.

E Set of fiber links, index e.

Demands and paths:

D Set of demands, index d. For each demand d, the tuple $<o_d, t_d, b_d>$ is given, where o_d and t_d are the origin and target nodes, and b_d is the bitrate in Gb/s.

P Set of pre-computed paths, index p.

$P(d)$ Subset of pre-computed paths for demand d. $|P(d)|=k, \forall d \in D$

r_{pe} Equal to 1 if path p uses link e.

Spectrum:

S Set of spectrum slices, index s.

$C(d)$ Set of pre-computed slots for demand d.

q_{cs} Equal to 1 if slot c uses slice s.

The decision variables are:

w_d Binary, equal to 1 if demand d cannot be served.

x_{dpc} Binary, equal to 1 if demand d is routed through path p and slot c.

Then, the LP-SA formulation is as follows:

$$\text{(LP-SA)} \quad \min \sum_{d \in D} b_d \cdot w_d \tag{2-10}$$

subject to:

$$\sum_{p \in P(d)} \sum_{c \in C(d)} x_{dpc} + w_d = 1, \quad \forall d \in D \tag{2-11}$$

$$\sum_{d \in D} \sum_{p \in P(d)} \sum_{c \in C(d)} r_{pe} \cdot q_{cs} \cdot x_{dpc} \leq 1, \quad \forall e \in E, s \in S \tag{2-12}$$

The objective function (2-10) minimises the amount of bitrate that cannot be served (rejected). Constraint (2-11) ensures that a lightpath is selected for each demand provided that the demand is served; otherwise, the demand cannot be served and therefore, is rejected. Constraint (2-12) guarantees that every slice in every link is assigned to one demand at most. Regarding size, the LP-SA formulation has $O(|D| \cdot k \cdot |C|)$ variables and $O(|E| \cdot |S| + |D|)$ constraints. It is worth mentioning that the RSA problem can be expressed as an extension of the *multicommodity flow problem* with additional constraints for the spectrum continuity and contiguity, therefore belonging to the NP-complete complexity class [It95].

2.4 Data analytics

Data analytics is a field devoted to inspecting, cleansing, transforming, and modelling large and heterogeneous amounts of data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making. Data analytics has multiple facets and approaches, encompassing diverse techniques under a variety of names that are applied in several fields.

2.4.1 Probability theory

A *random variable*, usually written X, is a function from an outcome space Ω taking values in a measurable space E, usually real-valued (2-13). In short, this mathematical definition allows defining a probability measure P in Ω expressing the likelihood that a measurable set of outcomes $A \subseteq \Omega$ (i.e. an event) happens. Therefore, a random variable itself does not return any probability; instead, these are given by the probability measure P. As an example, a random variable could be defined being Ω a population and X mapping a randomly selected person to his/her height, or to his/her number of children. In this cases, function P provides the

probability that a given person is within a given range of height or that has a given number of children.

$$X : \Omega \rightarrow \Re \tag{2-13}$$

When the image of X is finite or countably infinite, we say that X is a *discrete* random variable, and the distribution of event probabilities described by a *probability mass function P (pmf).* On the contrary, if the image of X is uncountable infinite we say that X is a *continuous* random variable and the distribution of event probabilities described by a *probability density function* (*pdf*). Any random variable can be described by its *cumulative distribution function* (*cdf*), which describes the probability that the random variable will be less than or equal to a certain value.

In the previous examples, a pdf would be used to compute the probability of being taller or shorter than a given value and a pmf used to compute the probability of having a certain amount of children. Either in the case of discrete or continuous random variables, we find a pre-defined set of *probability distributions* that can accurately model the distribution of events in many scenarios.

It is interesting to present the *normal* (or Gaussian) continuous probability distribution, whose pdf is defined in eq. (2-14). A normally distributed random variable X denoted as $X \sim N(\mu, \sigma^2)$, is characterised by a mean value μ and a standard deviation $\sigma \geq 0$. The normal distribution is commonly used to model many environments in several natural and social sciences, where events are likely to take values around a mean μ rather than distant to it. Fig. 2-14 shows a plot for the pdf of the normal distribution in eq. (2-14).

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2-14}$$

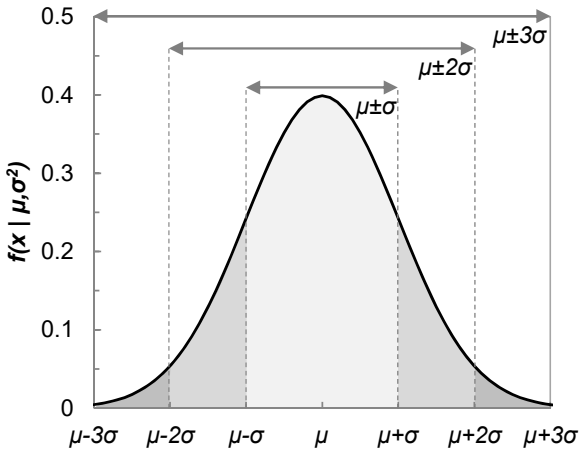

Fig. 2-14 Normal distribution probability density function.

As we can observe in the figure, the standard deviation provides some information about the probability of falling in a given region centred on the mean. Formally, the probability that a random sample of falls in the interval $I_k = [\mu - k \cdot \sigma, \mu + k \cdot \sigma]$ can be to configure confidence bounds. For example, for integer values k = 1, 2 and 3, the probability that a random sample will fall in the $\mu \pm k \cdot \sigma$ interval is 68.27%, 95.45% and 99.73%, respectively.

2.4.2 Statistic parameter estimation

Usually, we are interested in discovering information about the underlying probability distribution of empirical data presenting a random behaviour. For example, it could be desired to estimate the proportion of a population of voters who will vote for a particular candidate. That proportion (a probability in this case) is usually called *parameter,* and its estimation (i.e., the *estimate*) is based on an observed random sample of voters (i.e., the *sample*).

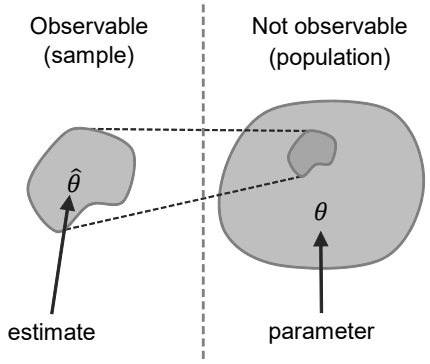

Fig. 2-15 Estimation of a parameter from a sample.

Since the estimate is computed based on particular observed data and not the entire *population*, the estimate will be an approximation of the parameter. These concepts are illustrated in Fig. 2-15.

Estimation theory is the branch of statistics that deals with estimating the values of parameters based on measured empirical data that has a random component. Among the different approaches that can be followed, we focus on the probabilistic approach which assumes that observed vectors of data $X=\{x_1,...,x_N\}$ are random with probability distribution dependent on the parameters of interest.

An estimate is said to be *unbiased* if its expectation equals the target parameter (eq. (2-15)). For illustrative purposes, equations (2-16) and (2-17) below can be used to compute unbiased estimates of for the mean and standard deviation μ and σ of a normally distributed data sample [Ho13]. The resulting values can then be compared to the original parameters (if known), using some error measure such as the absolute relative error, shown in eq. (2-18).

$$E(\hat{\theta}) = \theta \quad (2\text{-}15) \qquad \hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (2\text{-}16)$$

$$\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{\mu}) \quad (2\text{-}17) \qquad e = \frac{\left|\hat{\theta} - \theta\right|}{\left|\theta\right|} \quad (2\text{-}18)$$

2.4.3 Neural networks

Machine learning is a field in computer science devoted to program computers to use past data or experience to solve a given problem. Many successful applications of machine learning exist already applied in several fields, including economics, robotics, and bioinformatics. The field is closely related to other fields such as Mathematical Optimization and Computational Statistics. In data analytics, machine learning refers to the methods used to devise complex models and algorithms that lend themselves to prediction. Although machine learning encompasses a wide variety of fields, we focus our attention in *artificial neural networks* (ANN), since they play a central role in our proposed approach for predicting quality core network traffic. ANNs are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn how to do tasks by considering examples or past data, generally without task-specific programming. For example, they can be used to predict future values in a data stream, or to identify and classify patterns in images.

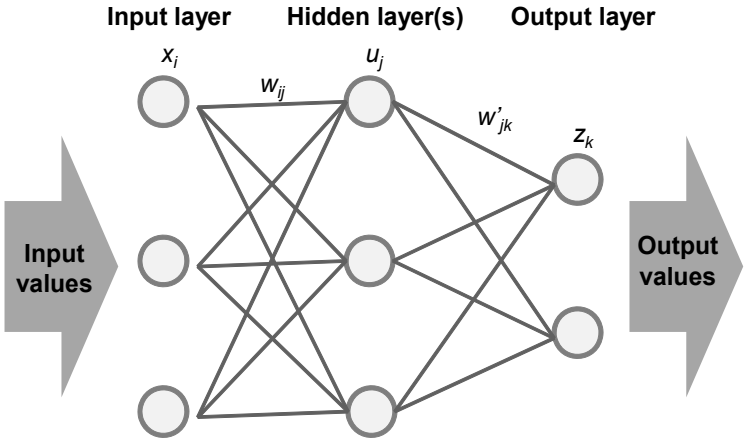

Fig. 2-16 Artificial neural network.

An ANN (Fig. 2-16) is based on a collection of connected units called *neurons* interconnected by a set of synapses. Typically, neurons are organized in *layers*, where each layer might perform different kinds of transformations on their inputs. Signals travel from the first (input) to the last (output) layer, possibly after traversing multiple hidden layers. A receiving neuron can process the signals from

preceeding neurons and then signal downstream neurons connected to it. Neurons may have a state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream. Further, they may have a threshold such that only if the aggregate signal is below (or above) that level the downstream signal is sent.

2.5 Conclusions

In this chapter, some background has been introduced necessary to understand the work presented in this thesis. First, optical transport networks have been introduced, reviewing basic concepts on transmission technology, network topology transponders and elastic optical networks. Next, multilayer networks have been introduced to understand how the VNT, a central element of this thesis, is defined.

In addition, a review of off-line network planning to state-of-the-art in-operation network planning has also been introduced to facilitate the reader the contribution of this thesis in the forthcoming chapters.

Tools for solving complex decision-making problems have been presented, paying special attention to mathematical programming and meta-heuristics for combinatorial optimisation problems. These tools are used to solve the combinatorial problems identified during the development of this thesis. The RSA problem has been finally presented to know the underlying subproblems present in our proposed algorithms and also to familiarise the reader with network optimisation problems.

Finally, some concepts related to the field of data analytics have been presented, gathering together probability and estimation theory as well as neural networks in machine learning. All these concepts are needed to understand properly the data analytics approaches presented in the following chapters.

Next chapter explores the state-of-the-art used as a reference point in the development of this thesis.

Chapter 3

Review of the State-of-the-Art

Next, we review the state-of-the-art related to the different goals targeted by this PhD thesis with the twofold objective of ensuring that these goals have not yet been covered in the literature and for serving as a starting point for work.

3.1 Traffic modelling

3.1.1 Data plane monitoring

Research on distributed monitoring architectures is recently receiving great interest. Authors in [Sa16] proposed a hierarchical monitoring architecture aiming at providing monitoring information gathering coming from different layers and network elements in a scalable way without overloading centralised controllers. Reactive strategies based on alarms pre-configured at different monitoring planes for several transmission parameters are proposed as the way to trigger network reconfiguration. This data analysis approach, however, results insufficient for detecting complex events such as traffic prediction. In fact, performing data analytics at the nodes requires from data stream mining algorithms [Me10].

Recently, authors in [AV17-2] proposed an early bit error rate (BER) degradation detection algorithm in the optical layer to meet the committed quality of service for demands in the MPLS layer. This approach evidences that extending data analytics to the network nodes across multiple layers also provides an increased knowledge about the network. To facilitate the introduction of multilayer data analytics, a monitoring architecture suitable for multilayer networks is required to provide mechanisms that allow grouping and correlating packet and optical monitoring.

Last but not least, CPU and memory are scarce resources in network nodes, which limits the number of *observation points* that can be configured. Therefore, huge research effort has been focused on making efficient use of them. For instance, since observation points can be configured not only in the source nodes of labeled switched path (LSP), but also in any node along their route, authors in [Mal14] proposed an algorithm that intelligently configures observation points and dedicates more monitoring resources to the most relevant flows while doing an aggregated monitoring of the rest, thus allowing more accurate traffic flow estimations. In light of this, monitoring architectures need to incorporate a flexible monitoring placement in addition to efficient use of node resources.

3.1.2 Traffic model estimation

Traffic modelling aims at finding mathematical models to represent the behaviour of traffic, also referred to as *teletraffic theory* [Wi98]. The first traffic modelling work was carried out by A. K. Erlang, where certain probability distributions for new calls characterized telephone traffic and call durations. Since then, teletraffic theory for packet networks has tremendously evolved by proposing new mathematical models to better model current network traffic [As14], being continuously challenged by new network technologies and services [5G] [Lee14].

Traffic modelling is not straightforward and can be done in multiple and interesting ways [Ch06]. Some proposals aim at obtaining mathematical models for both the packet size and the packet inter-arrival time (IAT). Whereas modeling packet size has been proven to be a simple problem, modelling the packet inter-arrival time remains a much more difficult one, for which a wide variety of solutions have been proposed including Pareto, Poisson and Weibull processes just to mention some [Zu03]. Another approach consists of finding a mathematical model describing the *packet ratio* (i.e., the bitrate) over time. Among these, Fractional Brownian motion and auto-regressive models (e.g. ARIMA) have been proposed in the literature to model network flow bitrate [Wei94]. Also, queuing theory models have been proposed to model not only traffic flows at the packet level, but also their processing at packet nodes [Zu03].

Packet traffic also varies depending on the network segment: traffic in access networks conveys few end-user connections and is bursty when compared to high capacity flows in metro and core segments, less bursty as a result of the aggregation of several connections [Ve13]. Given this, different works have been proposed to model traffic at different network segments [Lee14] [Ba04] [Hol03].

Another important aspect of traffic modelling relates to when the modelling occurs. *On-line* modelling occurs in real time while the network is in-operation. Streams of packet traffic are monitored until traffic models can be estimated based on the monitored data, thus allowing to use them in operation. This approach differs from *off-line* modelling, entirely based on history data with no real-time application. One

of the main difficulties of online modelling arises on big networks where a large amount of traffic flows needs to be modelled thus requiring huge storage and computing requirements [Zh03]. However, the evolution of cloud computing in the recent years allows efficiently storing and processing large and heterogeneous amounts of data, reason why large-scale on-line traffic modelling has received increasing attention [Gi16-1].

Because all of the above, the choice of the best traffic modelling approach is not immediate and depends on the specific context. In the context of this PhD thesis, we tackle the online modelling the bitrate of metro and core OD traffic flows based on monitoring the data plane, considering the current dynamicity of service layer traffic. Table 3-1 summarises the works surveyed above regarding traffic modelling. Although the literature contains works covering traffic modelling in its various aspects, to the best of our knowledge our proposal has not been studied. Although online data analytics is recently being applied to monitoring data [Gi16-1], most traffic modelling approaches are based on off-line traffic traces.

For that reason, goal G1.1 in this PhD thesis focuses on studying the best way to apply data analytics techniques on monitored traffic to obtain such models.

3.1.3 Traffic model re-estimation

Although traffic modelling techniques can be applied to achieve traffic prediction, sudden and unexpected changes in the monitored traffic can lead to inaccurate models. It can happen, for instance, when core-flow traffic patterns change as a result of changes in the injected traffic from adjacent metro network segments, e.g., when some metro-flows are redirected to a different entry node in the core. Although the detection of a change in a data stream has been extensively studied in the literature (commonly referred to as *concept drift detection* [Ga14]), new models need to be estimated after the original ones become no longer accurate. The re-estimation of new models entails long monitoring times (e.g. several weeks) during which no predictive models are available. Depending on the rate these changes re-estimation might not even be possible.

In light of this, new mechanisms need to be devised to ensure uninterrupted predictive capabilities in the core network. In this regard, some authors have proposed to monitor (small) individual flows instead of the (large) aggregated flows for traffic modelling purposes. For instance, authors in [Mal14] proposed a machine learning procedure that intelligently de-aggregates relevant monitored traffic flows and aggregates the rest to achieve accurate traffic estimations. However, this kind of selective flow monitoring approaches lacks the flexibility required to adapt OD traffic models against any potential metro-flow rerouting.

Another option is to apply data analytics in metro networks, based on monitoring data from metro nodes and estimate metro-flow traffic models. Metro-flow predictive models can be aggregated to re-estimate obsolete OD models in the core.

For instance, authors in [Yu09] proposed AutoRegressive Integrated Moving Average (ARIMA) for modelling and forecasting metro area network traffic flows as a function of application-layer traffic flow models in IP networks.

It is worth mentioning that the accuracy of predictive OD models cannot be based only on extensive metro-flow traffic monitoring. Although the literature offers a broad range of mathematical models for fitting flow traffic data, techniques for aggregating metro-flow models to guarantee reliable and accurate OD traffic predictions need to be explored. Among related works, the survey presented in [Lu11] reviews different approaches to predict time series of an aggregate using each flow forecast. Although model aggregation has been used in other fields (e.g., in bioinformatics [Ra09] or marketing [Ma08]), its application in the context of network traffic prediction is unexplored. For this reason, the contribution of goal G1.2 of this PhD thesis is to obtain core OD traffic models based on the aggregation of metro flow traffic models in real-time.

Table 3-1 summarises the different features studied in the literature related to some aspect of traffic modelling.

Table 3-1: Survey of traffic modelling

Feature	References
Reactive monitoring architecture	[Sa16]
Access network traffic modelling	[Lee14], [As14]
Metro and core network traffic modeling	[Ba04], [Zh03], [Hol03], [Yu09]
Packet-wise modeling	[Ch06], [Zu03], [Ba04]
Flow-wise modeling (i.e., bitrate)	[Ch06], [Ba04], [Zh03]
Traffic flow aggregation	[Mal14]
Model inaccuracy detection	[Ga14]
Predictive model aggregation	[Lu11], [Ra09], [Ma08]

3.2 VNT reconfiguration

3.2.1 VNT reconfiguration approaches

Because of its benefits, VNT reconfiguration has been widely studied in the literature. Authors in [Agr09] proposed a centralised path reallocation module running periodically aiming at minimising the number of used transponders. The approach, named as Optical Resource Optimization (ORO) is triggered on a fixed basis to minimize the number of used transponders of the VNT. Although the good

results obtained by running the ORO algorithm, its execution is triggered at fixed time periods rather than based on some performance metric (e.g., the blocking probability). This unawareness of the network performance might bring performance issues when considering dynamic network traffic scenarios.

To alleviate this problem and follow traffic changes, authors in [Gen03] proposed to reconfigure the VNT according to vlink utilisation. By setting low and high utilisation thresholds (watermarks) on each vlink's capacity, VNT reconfiguration is triggered whenever one of these thresholds is violated, by either removing or adding a vlink, respectively. Although this VNT reconfiguration approach adjusts the capacity of the VNT accordingly to traffic changes, its scope is local (per vlink), and thus it might bring sub-optimal savings when compared to global-scope VNT reconfiguration. For this very reason, this approach lacks a wider vision of the traffic evolution, such as end-to-end (OD) traffic variations.

3.2.2 VNT reconfiguration based on traffic prediction

In light of the above, further research has been carried out aiming at achieving more knowledge about traffic changes. For instance, by using monitored data to produce estimations that can help to anticipate changes in the traffic and proactively reconfiguring the VNT beforehand. In this regard, authors in [Oh10] proposed a method for reducing errors in traffic estimations by adapting the VNT reconfiguration result at multiple stages over time. Although this approach shows the benefits of performing a global VNT reconfiguration based on predicted traffic matrices, authors do not provide a means to obtain such prediction from the network while it is in-operation. The same problem arises in a different work [Fe15], where a decision-making system to select pre-defined solutions for the VNT based on predicted traffic matrices is used rather than an on-line VNT reconfiguration algorithm. Finally, authors in [Ve16] used future traffic estimation to trigger a VNT reconfiguration after the detection of an anomalous traffic increment between two network nodes.

A very similar problem, known as the *multi-hour* VNT reconfiguration problem, has been studied in the literature [Ap10] [Ap12] [Pa10]. In this problem, the VNT is designed off-line for a set of n periods (e.g. the days of the week) using n predicted traffic matrices. The predicted traffic matrices are used as input of an optimisation problem that outputs the sequence of VNT for each period minimising OPEX in terms of used transponders and optical spectrum. Although the multi-hour problem includes traffic prediction in the process of VNT reconfiguration, it is solved off-line before the n considered periods, which might entail inefficiencies under dynamic traffic scenarios where OD traffic patterns can suddenly change. For example, assuming a reconfiguration at time t based on n predicted traffic matrices for times t+1, ..., t+n, an unexpected OD traffic change between t and t+n could cause inefficiencies in the solution, since some of those traffic matrices would

be no longer representative of the actual traffic. In those scenarios, a single-period, on-line VNT reconfiguration approach would prefered to reconfigure the VNT based on a more reliable, short-term traffic prediction. In addition, this sort of VNT reconfiguration would require from powerful algorithms to analyse large amounts of traffic monitoring data in real time to anticipate traffic changes when possible. For instance, a data analytics algorithm could run periodically (e.g., every hour) to predict traffic, and, in case the VNT needed to be reconfigured trigger the execution of a VNT optimiser to find the optimal topology for the next period.

3.2.3 Solving methods for VNT reconfiguration

Let us finally focus on the solving methods used to solve the VNT reconfiguration problems presented above. In [Agr09], the ORO algorithm is formulated as an ILP problem, and, in light of its complexity for realistic instances, a GRASP-based heuristic algorithm is proposed to obtain quality solutions in practical times. Similarly, an ILP formulation along with a heuristic algorithm are presented in [Gen03]. This time, however, the heuristic algorithm is ad-hoc and is not based on any metaheuristic framework. Instead, it is based on two major stages: a vlink addition stage followed by a vlink removal stage. The heuristic algorithm presented in [Oh10] is also based on these two stages, but performing them globally based on an estimated traffic matrix. Finally, the solving method proposed by authors in [Fe15] is not based on an on-line VNT reconfiguration algorithm. Instead, a pool of VNT are generated off-line, based on a VNT design algorithm (named as CONGA-VTD). Once in-operation, a decision-making system decides which of the pre-computed VNT adapts better the predicted traffic matrix for the next period.

Table 3-2 summarises the works surveyed above regarding VNT reconfiguration. Although the literature contains similar works, to the best of our knowledge there is not a previous one featuring a single-period, on-line, global VNT reconfiguration based on traffic prediction considering the increasing dynamicity of core OD traffic. For this reason, the contribution of goals G2.1 and G2.2 of this PhD thesis is to devise, implement and evaluate a VNT reconfiguration approach gathering all these features at the same time.

Table 3-2: Survey of VNT reconfiguration

Feature	References
Global VNT reconfiguration	[Agr09], [Fe15]
VNT reconfiguration based on performance metrics	[Gen03], [Oh10], [Ve16]
VNT reconfiguration based on predicted traffic matrices	[Oh10], [Fe15], [Ve16], [Ap10] [Ap12] [Pa10]
VNT reconfiguration under dynamic traffic	[Ve16]

3.3 Metro-core traffic orchestration

Different works have addressed the topic of metro-core orchestration. Regarding path computation across multiple network domains, authors in [Ca16] proposed a broker-plane above the management planes of network domains. By allowing the broker to obtain available resources from each network domain, end-to-end resource management and multi-domain path provisioning networks is enabled with reduced blocking probability. In a more recent work [Pro18] authors proposed to add cognition to the broker by collecting per-domain quality of transmission (QoT) monitoring data from the optical layer to assist the end-to-end multi-domain RMSA process. Although this architecture facilitates end-to-end provisioning across multiple domains based on traffic monitoring, the information about traffic flows from the MPLS layer remains unshared between adjacent network domains and hence the possibility of re-optimising the core VNT based on metro-flow traffic information.

In that regard, authors in [Al16] addressed the impact of human mobility patterns of mobile users in a metro area over the aggregated traffic offered to the mobile metro-core networks. By exploiting predictable trends recognised in aggregated core traffic, they propose two optimisation methods to reduce the energy consumption in the optical layer of metro-core networks. Although this work proposes to predict core flow traffic based ultimately on monitoring data from metro networks, such predictive models are obtained from off-line metro network history data. This off-line traffic prediction approach might result inefficient when applied to a changing metro traffic scenario, where the re-routing of some metro-flows to a different entry node in the core would change the aggregated traffic patterns in the core thus dropping the predictive quality of the original models. To alleviate this problem, it would be advisable to enhance the previous approach to an online one, where the aggregated core traffic could be predicted based on online, metro-flow traffic monitoring. For this reason, the contribution of goal G1.2 and G2.3 in this PhD thesis is to efficiently keep up-to-date predictive OD traffic models for VNT reconfiguration under dynamic traffic, by aggregating several metro network flow traffic models from neighbouring metro network areas.

Table 3-3: Survey of metro-core orchestration

Feature	References
End-to-end service orchestration in multi-domain networks.	[Ca16], [Pro18]
Core network optimisation based on metro traffic patterns.	[Al16]

3.4 Conclusions

In this chapter, we have reviewed the state-of-the-art of relevant works related to the goals of this thesis.

Regarding traffic modelling, several works have been proposed to model traffic, including specific ones studying traffic modelling in metro and core networks. Nevertheless, there are not any works devoted to online traffic modelling of the OD bitrate in metro and core networks, assuming the current dynamicity of service layer traffic. It is also of special interest to guarantee uninterrupted traffic predictive capabilities in the core network under changing traffic scenarios. Although several works have studied concept drift detection, the re-estimation of new models entails long monitoring data collection times (e.g., several weeks). Therefore, new mechanisms new to be explored to achieve real-time predictive core traffic model re-estimation.

The literature contains multiple works regarding VNT reconfiguration, presenting partial or global reconfiguration approaches, some of them based on estimated traffic matrices and some triggered by a decision-making system. However, none of them presents a reconfiguration approach jointly featuring online, global VNT reconfiguration based on traffic prediction. The approach above will be studied in this thesis.

Finally, although some works focusing on metro-core orchestration provide a means for efficient end-to-end resource allocation across multiple domains, the proposed architectures lack mechanisms to allow the exchange of traffic-flow related information between the participating network domains. Other approaches consider such exchange of information in the form of predictive traffic models for the core based on metro-flow traffic data. Although these models are used for efficiently re-optimising the core, they are built on an off-line basis and therefore lack of re-estimation mechanisms to ensure quality traffic prediction under changing traffic. For this reason, the approach above will also be studied in this thesis.

Several open issues have been detected. Next chapters will focus on presenting our contributions for each of these unexplored issues that will eventually lead to the consecution of the main goal of this thesis.

Chapter 4

Simulation environment

Studies involving multilayer networks, in particular, those dealing with in-operation network planning may need of evaluation in complex scenarios. The access to test-beds providing realistic scenarios is usually limited, which demands alternative evaluation methods for the research community.

In that regard, network simulation has been proven as a useful tool. It does not only overcome the limited access to physical resources but also offers a means of implementing and evaluating algorithms in an accessible and controlled environment, so they can be verified and improved before being brought into practice hence increasing the chances of success.

Although previous work has been done regarding multilayer network simulation [CaPhd], it does not entirely match the requirements needed to properly evaluate the algorithms and network scenarios presented in this thesis. For that reason, we present a new multilayer network simulation environment for cognitive in-operation planning: iONESim. It is based on the OMNeT++ simulation framework [OMNeT] widely used among the research community.

4.1 Simulator overview

In this section, we present the modules defining the structure of iONESim as well as their interconnections. Although these definitions will remain constant for all simulation scenarios, some of their parameters can be modified to simulate different network scenarios.

Fig. 4-1 shows how iONESim modules communicate with each other. According to the OMNeT++ simulation framework, each iONESim module is attached to a C++

class and exchanges data with different modules either in terms of the OMNeT++ built-in messaging system or by making API calls to public class methods. In the latter case, a pointer to the desired module can be obtained before the simulation starts (i.e., at initialisation time).

There are three main module groups in iONESim modules: data plane modules, control plane modules and event modules. Data plane modules are responsible for the simulation of the network data plane, featuring a traffic generation framework able of generating OD traffic either for metro or for core network flows during the simulation, according to customizable service layer traffic patterns. Control plane modules simulate the role of a network controller, including an SDN controller, a VNT manager (VNTM), centralised data analytics and statistic collection module for post-simulation analysis. Finally, event modules trigger external events that have an impact in the network performance, such as metro traffic re-routing or just the management of the simulation timing.

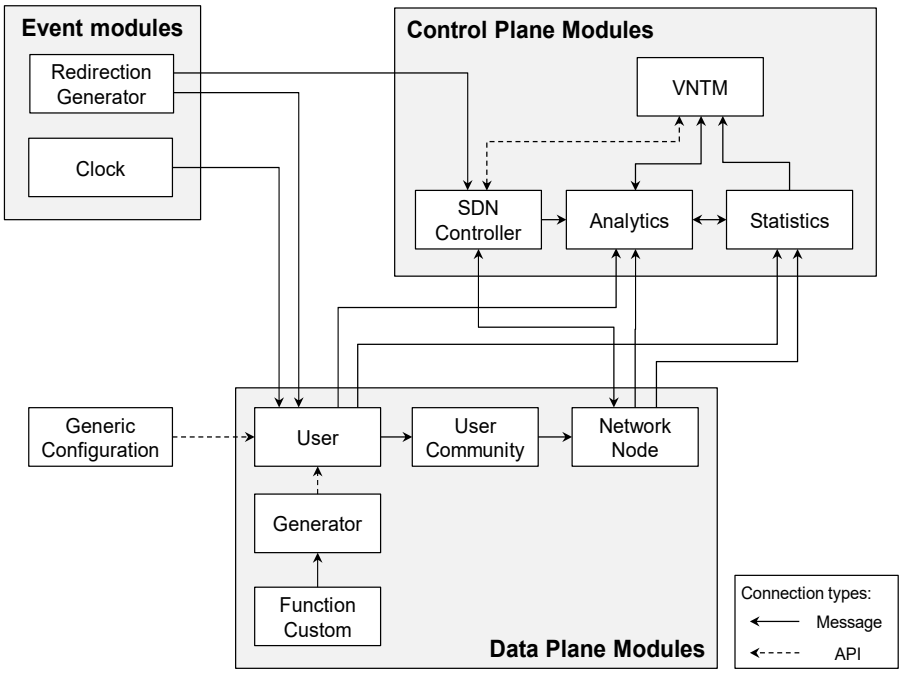

Fig. 4-1: iONESim modules and their interconnection.

4.2 Data plane modules

Taking into account the evolution of service traffic in the recent years, we aim at simulating OD traffic patterns matching an evolving traffic scenario. To that end, a traffic generation framework has been devised as a core feature of iONESim offering not only a flexible way to simulate network traffic flow data but also as a

generic data stream generator. The approach followed in this thesis is to periodically generate flow data (i.e., bitrate) for all the OD pairs in the VNT (i.e., a periodic update of the traffic matrix). This is in contrast with other approaches followed in previous simulators [CaPhd], where a series of *demands* arrive to the network following a Poisson-based stochastic process. Therefore, there are not demands in iONESim. Instead, we consider time-varying OD bitrate flows.

To generate customizable traffic patterns either for core or for metro network flow traffic, we devised a series of modules with independent functionalities that all together allow the periodic generation of OD traffic matrices. The data plane modules responsible for the traffic generation are *FunctionCustom* and *Generator* whereas the *User*, *UserCommunity* and *NetworkNode* modules are responsible for the aggregation and injection of the generated traffic matrices into the VNT.

4.2.1 Function Custom

To allow a high degree of customisation in the generation of traffic, FunctionCustom modules are devised to facilitate generating time-dependent, correlated real-valued data. The main function of a FunctionCustom module is mathematically stated in eq. (4-1), where variable t represents the current simulation time point and vector v_t an ordered tuple representing the n previous values of f (eq. (4-2)), allowing introducing autocorrelation in the data.

$$\begin{aligned} f &: \Re \times \Re^n \to \Re \\ (t, \vec{v}_t) &\mapsto v = f(t, \vec{v}_t) \end{aligned} \tag{4-1}$$

$$\vec{v}_t = \left[f(t-1, \vec{v}_{t-1}), \ldots, f(t-n, \vec{v}_{t-n}) \right] \tag{4-2}$$

Although new FunctionCustom modules can be implemented to represent the different components of the traffic, we can identify that these functions usually result from the composition of simpler mathematical functions. To ensure re-usable and well-structured code, a library of non-module classes is defined to facilitate the composition of new FunctionCustom modules. These simpler functions are classified into two main categories, namely *FunctionUnitary* or *FunctionDistribution*. A FunctionUnitary implements a real-valued function defined in the [0,1]×[0,1] square. This domain and range constraints are imposed in order to avoid the duplication of essentially the same mathematical procedure. Examples of these are polynomial and periodic functions. On the other hand, FunctionDistributions are devised to implement random variables, based on pseudo-random real number generation of given probability distributions. Examples include the normal, the chi-squared and the uniform distribution.

4.2.2 Generator

Although the definition of FunctionCustom is flexible enough to allow the implementation of an entire OD traffic profile, an upper aggregation level is defined by the Generator module to allow combining different FunctionCustom modules.

Formally, the Generator module combines a set of FunctionCustom modules providing the different components defining a traffic profile. This combination of FunctionCustom modules is done by evaluating an arithmetic expression which is parsed at initialisation time and later evaluated every time that a traffic (bitrate) generation request is triggered. Multiple Generator modules might be used within a simulation in order to generate different OD traffic profiles using various different FunctionCustom. For example, two Generators could be defined to inject user-related traffic (e.g. video) or datacenter-related traffic (e.g. database synchronisation) into the VNT.

The Generator modules provide the endpoint for the OD bitrate generation. Next, the modules responsible for aggregating and injecting this traffic into the network are presented in detail.

4.2.3 User

One of the goals of this thesis is devoted to metro traffic flow modelling. To numerically assess our proposals, iONESim must include a means of working with metro-flow traffic. For that reason, the User module is defined to implement the generation of multiple metro-sourced traffic flows traversing the core VNT. In particular, it contains information about the ingress node in the core VNT, the set of egress nodes in the core VNT (defining one metro-flow each) and one Generator module providing the same traffic profile for the User metro-flows. Thousands of User (hence metro-flows) are configured in iONESim simulations to analyze our proposed algorithms in realistic traffic scenarios properly.

Although the ingress node of a User module is set at initialisation time, we advance that this can be modified once the simulation has started by the RedirectionGenerator module. In contrast, the destination nodes and the associated Generator remain constant during the simulation.

Once a User module receives a traffic generation request, it generates a new bitrate value by calling the corresponding Generator module. This new value is bundled with ingress and egress information and sent to its parent UserCommunity module. Also, this data is also sent to the Analytics module to simulate metro-flow traffic monitoring.

4.2.4 User Community

The UserCommunity module is defined to act as an aggregation point for User-generated traffic; the module aggregates all the User-generated traffic matrices entering the core VNT through the same ingress node. For that reason, the simulation must contain as many UserCommunity modules as edge VNT nodes.

The single role of a UserCommunity module is to periodically aggregate all the received metro-flow traffic into a single traffic matrix and eventually send it to the associated ingress node. It is worth mentioning that the matrix built by a UserCommunity is only one row of the complete core OD traffic matrix since all the OD pairs are sourced in the same node.

4.2.5 Network Node

So far, we have presented the traffic generation framework formed by the FunctionCustom and the Generator modules, which can be configured to shape different traffic patterns for metro-flow traffic. User modules representing metro traffic flows use this framework, periodically calling the Generator modules to generate new metro-flow traffic matrices with ingress and egress nodes in the core VNT. Finally, UserCommunities aggregate bitrate requests from several Users composing one row of the core OD traffic matrix. To process the allocation of resources in the VNT according to the new traffic matrices, we need to provide a meeting point between the data and the control plane modules. This is accomplished by the NetworkNode module, a high-level representation of an edge MPLS router whose main function is to notify traffic matrix updates to the control plane. An IP address uniquely identifies every NetworkNode module.

At regular intervals each NetworkNode receives an updated traffic matrix from its associated UserCommunity; upon reception, a request containing the updated traffic matrix is sent to the control plane aiming at (*i*) allocating the requested capacity in the VNT and (*ii*) simulating OD core traffic monitoring. Data plane simulation ends at this point giving way to control plane modules, presented next.

4.3 Control plane modules

The main contribution of this thesis is the combination of data analytics with optimisation techniques in order to apply the OAA loop in the network. This contribution is presented in the following chapters, using iONESim to assess their performance. We advance the reader that our proposal is based on a centralised network architectures scheme, and therefore a series of control plane modules are defined in iONESim.

4.3.1 SDNController

The SDNController module is responsible for processing the (partial) OD traffic matrices sent by the NetworkNodes modules during a simulation. It stores a representation of the core VNT, the allocated paths and the current traffic matrix resulting from combining all the received sourced (row) traffic matrices and operates these three elements according to the changes in the traffic.

At initialisation time, the module sets up the initial VNT, the routes and the traffic matrix from an external file. Once a simulation starts, the SDNController is the only responsible for operating these three elements, with the only exception of the Virtual Network Topology Manager (VNTM) to whom management can be promptly delegated for cognitive running in-operation planning algorithms.

OD bitrate requests received from NetworkNodes are processed at the SDNController by running a routing algorithm based on general or per-OD pair routing policies. Routing policies are defined to establish resource-related constrains in the allocation of requested OD bitrate. Once a traffic matrix from a NetworkNode has been completely processed, a response message is sent back to the original NetworkNode containing information about how the request was processed (e.g., the amount of served unserved traffic for each OD pair and the paths supporting the served capacity).

4.3.2 Analytics

The combination of data analytics with optimisation techniques is the central part of this thesis, aiming at applying the OAA loop in the network. To enable data-analytics based simulation, iONESim includes the Analytics module which allows centralized data analytics based on monitoring data. Despite being a single module, it is composed of a family of submodules that provide different data analytics –related functionalities and that are orchestrated during the simulation. The Analytics module implements one of the major contributions of this thesis, which is the data analytics framework presented in Chapter 5. Hence, we refer the reader to that chapter for further details.

As already mentioned, every User and NetworkNode in the simulation periodically sends their generated traffic matrices to the Analytics module. This is done to simulate traffic monitoring capabilities both for metro-flow and core OD traffic.

Every time that new monitoring data is received, it is stored in a BigDataRepo (Fig. 4-2) simulating a noSQL database [Gi16-1]; hence, monitoring data is stored as new bitrate values are generated. At a configurable period, the traffic samples in the BigDataRepo are processed and transformed into modelled data and stored in a ModeledDataRepo; modelled data summarises monitoring data into relevant

statistic features. This procedure is periodically applied thus building monitoring time series during the simulation.

Once enough monitoring data is available in the repositories, an Estimator module might trigger the estimation of traffic models capable of predicting metro flow or core OD traffic (traffic model estimation is studied in detail in Chapter 5). Once obtained, traffic models are stored in a PredictiveModelRepo where they will be available for use and periodically evaluated by an Estimator module, aiming at ensuring a target predictive quality along time. The predictive models stored in the repository are used to serve traffic predictions to other simulation modules.

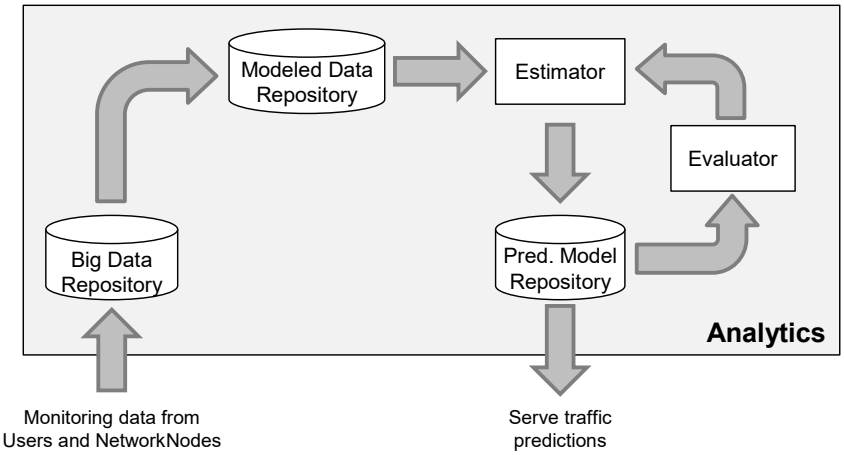

Fig. 4-2: Structure of the Analytics module in iONESim.

4.3.3 VNTM

In iONESim, the Virtual Network Topology Manager (VNTM) module is responsible for the execution of cognitive in-operation network planning algorithms that ensure a good performance in the core VNT. Although the SDNController is in control of the VNT during the simulation, it can delegate its management to the VNTM temporarily whenever it detects a performance issue in the VNT. As it will be explained in the forthcoming chapters, cognition is added to in-operation planning by combining the Analytics and the VNTM modules.

It is worth mentioning that the VNTM is designed to be easily extended by implementing new VNTM algorithms which can be enabled or disabled at choice for a particular simulation.

4.3.4 Statistics

The Statistics module is responsible for the collection of different statistics during a simulation. To implement new statistics more flexible, a modular approach has

been followed, where multiple containers are defined to collect and produce a different kind of statistic data. The choice of these containers is customizable and is done before the simulation starts. Examples of statistics containers are blocking probability computation, transponder utilisation or monitoring traffic data visualisation.

As a result of this, the Statistics module acts as an intermediary between its own containers and the rest of simulation modules, facilitating to the formers relevant data for statistics computation which is made available by the latter (e.g., the served or unserved OD bitrate is used to compute blocking probabilities).

4.4 Event modules

4.4.1 Generic Configuration

The GenericConfiguration module sets up the traffic generation specified by the User at initialisation time. It reads an input file defining the attributes characterising the traffic requests of each User (i.e., their ingress and egress nodes and the Generator providing bitrate values).

Besides that, it is also responsible for miscellaneous tasks facilitating the control of the simulation such as pseudo-random number generation and logging.

4.4.2 Redirection Generator

The RedirectionGenerator module is responsible for modifying the ingress node of a given set of Users during a simulation. In order to do that, the module parses an external file before the simulation begins containing information about the redirections (i.e., affected Users, new ingress nodes and simulation time when the redirections take place). Once parsed, these redirections are scheduled and eventually notified to the affected User modules once they are triggered. Every time that a notification is received by a User, this proceeds to switch to the specified UserCommunity, with the corresponding change in the core OD traffic patterns.

4.4.3 Clock

In iONESim, simulation timing is controlled by the Clock module, running a configurable periodic timer (*tick*). Every elapsed tick the module broadcasts a traffic generation request to all User modules, effectively starting a new wave of metro-flow bitrate value generation that will eventually produce partial (row) updates in the core OD traffic matrix. Each of these updates is processed by the

SDNController and by the Analytics modules, to adjust the allocated capacity in the VNT and to apply data analytics based on monitoring data, respectively.

The Clock tick is customizable and is typically set to a small value (e.g., one minute). The choice of this value has an impact on different aspects of the simulation that are worth noticing. If a small Clock tick is configured, more traffic generation waves will be triggered. This will slow down the simulation as a result of an increased number of events to be processed. Also, memory consumption will increase as a result of an increased granularity in the monitoring data (more data being stored). However, increased granularity in the monitored traffic might improve the predictive model quality. Conversely, if a large Clock tick is used less traffic generation, waves will be triggered thus speeding up the simulation and decreasing memory consumption. However, the quality of the predictive models might drop due to a lower granularity in the monitoring data.

4.5 Simulation workflows

In this section, the main simulation workflows that are executed in iONESim during a simulation are presented.

4.5.1 Traffic generation

The first workflow in this section shows the messages exchanged aiming at generating new traffic matrices for every User module. This is a core workflow since it allows the continuous generation of traffic matrices during all the simulation. Recall that the generation of traffic in iONESim is scheduled periodically by the tick value of the Clock module. Therefore, this workflow is started every Clock tick.

Fig. 4-3 shows the workflow in detail. Every time that a Clock period is elapsed (1), the Clock module sends a new message to every User module in the simulation at the same simulation time (2). These messages notify each User module that a new traffic matrix must be generated. When a User module receives such message, it requests new bitrate values to its related Generator module (3) in order to produce a new metro-flow traffic matrix; note that the same Generator will be called as many times as the number of destinations (i.e., metro-flows) of that User.

Every time that a new bitrate value is requested to a Generator module, this evaluates an internal arithmetic expression combining its child FunctionCustom modules. Because of this, during the expression evaluation, multiple calls to different FunctionCustom might be done (4). The final result of evaluating that expression (i.e. a new bitrate value) is eventually returned to the User module (5).

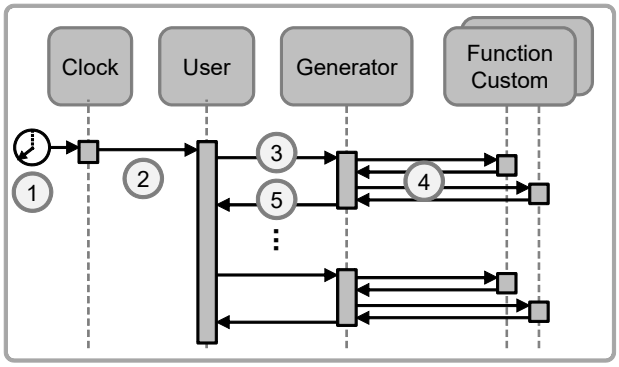

Fig. 4-3: Traffic generation workflow.

4.5.2 Traffic request

Fig. 4-4 presents the workflow continuing with the previous one, right after each User module has built a new metro-flow traffic matrix. In consequence, this workflow is also triggered every time that a Clock period is elapsed.

A User module performs two main actions after building a new traffic matrix. In one hand, a new traffic matrix is sent to the Analytics module (1) to simulate metro-flow monitoring. On the other hand, the matrix is sent to its parent UserCommunity to inject the traffic into theVNT (2).

When a UserCommunity module receives a new metro-flow traffic matrix from one of its child Users, it aggregates it with the rest of received metro-flow traffic matrices in order to produce a final, core OD traffic matrix for its associated NetworkNode. Since the traffic matrix generation happens at the same simulation time for all Users, it would be inefficient to forward the OD traffic matrix to the NetworkNode every time a User module updates it. To prevent this, each UserCommunity stores a temporary counter storing all the remaining User modules yet to send their matrices. This counter is reset every Clock period, thus allowing the UserCommunity to keep track of the remaining updates left. When the count reaches zero, the updated traffic matrix is finally sent to the NetworkNode module (3). Recall that the core OD traffic matrix contains no longer the disaggregated metro-flow bitrate information, but its aggregation (OD bitrate).

Once a traffic matrix from a UserCommunity is received, a NetworkNode executes two main actions. First, it updates its internal traffic matrix and sends a copy to the Analytics module (4) to simulate core OD traffic monitoring. Secondly, it sends a copy of the matrix to the SDNController module requesting the allocation of resources to serve the updated bitrate entries in the matrix (5). The SDNController finally adjusts the allocated capacity for each OD pair in the VNT according to the new values and sends back a response to the NetworkNode (6).

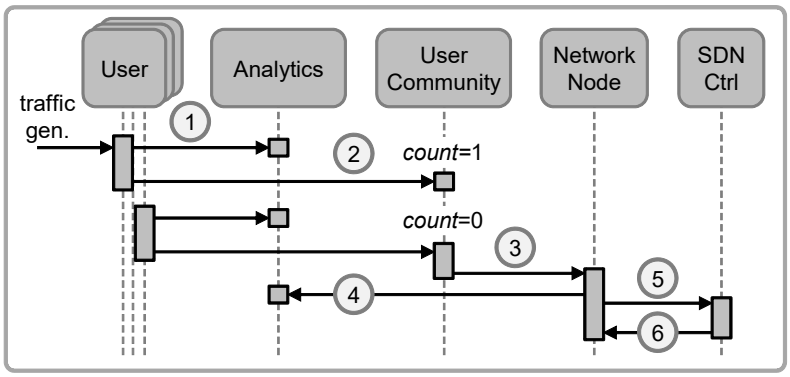

Fig. 4-4: Traffic request workflow.

4.5.3 Traffic modeling

As we have seen in the two previous workflows, either metro-flow or core OD traffic matries are sent to the Analytics module, simulating metro and core traffic monitoring. The workflow in Fig. 4-5 shows the interaction between the different Analytics submodules to store and process this data, simulating the process of data analytics based on monitoring data. The workflow only shows the messages exchanged between the different Analytics submodules; for details about the actual data analytics procedures taking place in this workflow we refer the reader to Chapter 5.

When new monitoring data (either metro or core) is received, it is stored in the big data repository (1). The monitoring data remains stored until a configurable periodic timer is elapsed. Once every period (2) monitoring data is conveniently processed from the big data repository and summarized into modelled data which is stored in the modelled data repository (3).

The data stored in the modelled data repository can feed various data analytics - based procedures, among which we find predictive model estimation. When the Estimator module checks that enough monitoring data is available to produce quality traffic models, the corresponding estimation is triggered (4). The estimator module then queries the modelled data repository the monitoring data needed for the estimation of the new metro-flow or core OD models (5). The new models are finally stored in the predictive model repository (6).

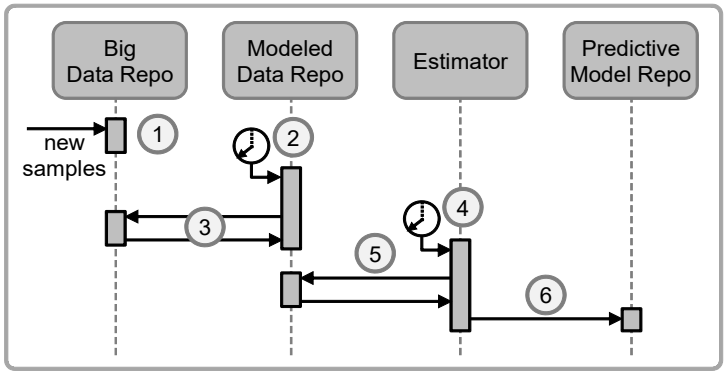

Fig. 4-5: Data analytics workflow.

4.5.4 VNTM algorithms

In the previous workflow, we have seen the role and interaction of the Analytics submodules, starting from monitoring metro and core flow traffic to finally producing predictive models. These models can be used to apply the OAA loop in the network. Effectively, by combining data analytics (Analytics module) with network optimisation (VNTM), we enable cognitive in-operation planning.

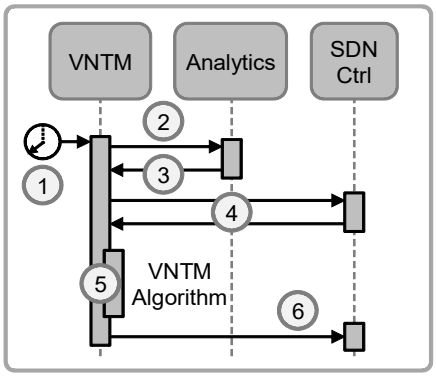

Fig. 4-6: Example workflow for cognitive in-operation network planning.

Fig. 4-6 shows an example workflow executed periodically to re-configure the VNT. Periodically (e.g., every hour) a timer is triggered in the VNTM (1) to run a VNTM algorithm based on data analytics, for example involving traffic prediction. Next, the VNTM module requests a traffic matrix prediction to the Analytics module (2) that is computed and returned in a reply message (3). After receiving the traffic prediction, the VNTM module requests the temporal management of the VNT to the SDNController module, thus completing all the input data needed to run the VNTM algorithm. The algorithm is executed (5) and its solution eventually implemented in the VNT (6).

4.5.5 Metro-flow rerouting

The last workflow in this section details the process of redirecting some metro-flows towards different ingress nodes in the VNT. This is achieved by modifying the parent UserCommunity of a given set of User modules. Redirecting metro-flow traffic to a different ingress node triggers changes in the core OD traffic patterns; this, in turn, might affect the predictive quality of the OD traffic models and also the performance of the VNT. This thesis devotes Chapter 8 to analyse this network scenario.

The workflow (shown in Fig. 4-7) starts whenever a redirection scheduled by the RedirectionGenerator module is executed (1). Initially, each affected User module is notified to change to a different ingress node in VNT (2). To that end, a User proceeds to switch to a different UserCommunity as follows: first, a message is sent to its current UserCommunity to notify unsubscription (3); as a result, the UserCommunity will decrease the amount of subscribed User by one. Once unsubscribed, the User module will subscribe to the new UserCommunity (4), increasing in the latter the User subscriber count by one. In the following traffic generation waves, traffic matrices will be sent to the new UserCommunity.

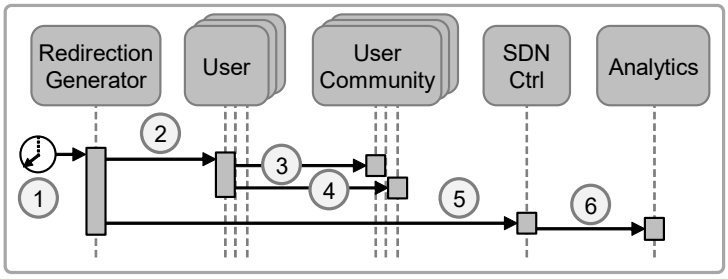

Fig. 4-7: Workflow for metro-flow ingress node redirection.

After all the affected User modules are redirected to their new ingress nodes, the RedirectionGenerator module notifies the whole redirection event to the SDNController module (5). The SDNController then computes the set of affected OD pairs by inspecting the set of rerouted metro-flows and sends such information to the Analytics module (6). This last step is done to allow the deletion of obsolete monitoring data and inaccurate predictive models for the affected OD pairs, thus re-starting the process of traffic modelling.

4.6 Conclusions

In this chapter, the iONESim network simulator has been introduced. We devised it as a means of evaluating the algorithms for cognitive in-operation network planning proposed in this thesis. According to the specific goals of this thesis, the simulator allows combining in-operation network planning with data analytics

based on monitoring the data plane. The simulation of metro and core traffic flow data is based on a complex traffic generation framework.

Chapter 5

Network traffic modelling

In this chapter, we study modelling techniques for network traffic flows. This is, to obtain a mathematical model describing the bitrate from a network node to some other destination at a given time t. To that end, we first devise a centralised data analytics framework which is located in the network controller. Through a series of data analytics modules, the framework collects monitoring data from packet nodes, processes it into meaningful modeled data and finally estimates predictive models based on this data. Predictive models for each monitored traffic flow are finally stored for their later utilisation.

Aiming at obtaining quality predictive models, we study in detail two different model estimation approaches, each one providing a series of trade-offs under different applications. Both approaches are introduced, formally explained and eventually evaluated through simulation.

5.1 Traffic flow modelling

5.1.1 Data analytics modules for traffic flow modelling

Let us assume a scenario where MPLS routers are extended with traffic monitoring capabilities thus allowing the obtention of bitrate samples. Each bitrate sample $x \in X$ is obtained following the node architecture in [Gi17]: initially, bitrate is monitored at packet nodes where bit counters are continuously updated during short *granularity periods* of duration G (e.g., every 1 minute); once a period G ends, counters are processed to produce a new bitrate measurement. To alleviate the number of samples sent to the network controller, these are aggregated in the router by computing the arithmetic mean in a larger *monitoring period* T (e.g.,

every 15 minutes) hence producing a single sample *x*=<*time*, *bitrate*>. Monitoring samples are sent to a collected data repository in the central controller storing a time series *X* for each monitored traffic flow, as illustrated in Fig. 5-1.

At regular intervals, collected data of a traffic flow can be conveniently summarized by applying data stream mining techniques, hence converting a time series *X* in a single modelled data value *y*. Modeled data includes, among others, the minimum, maximum, average and last collected bitrate value within some period (e.g., every hour). Every time that a newly modelled data *y* is generated, it is added to a modelled time series *Y* of the corresponding traffic flow, which is stored in a modelled data repository. It is possible to keep *Y*=*X* if no summarisation of the data is needed.

To obtain predictive traffic models, an estimator module periodically queries the modelled data repository to check if the amount of modelled data (i.e., the length of *Y)* is large enough to ensure a target predictive quality, and if so, proceeds to the execution of an estimation algorithm. The estimation algorithm receives as input a modelled time series *Y* and returns a mathematical model predicting the modelled variable at any time (e.g., the maximum bitrate). Once a predictive model has been computed, it is stored in a predictive model repository for future utilisation.

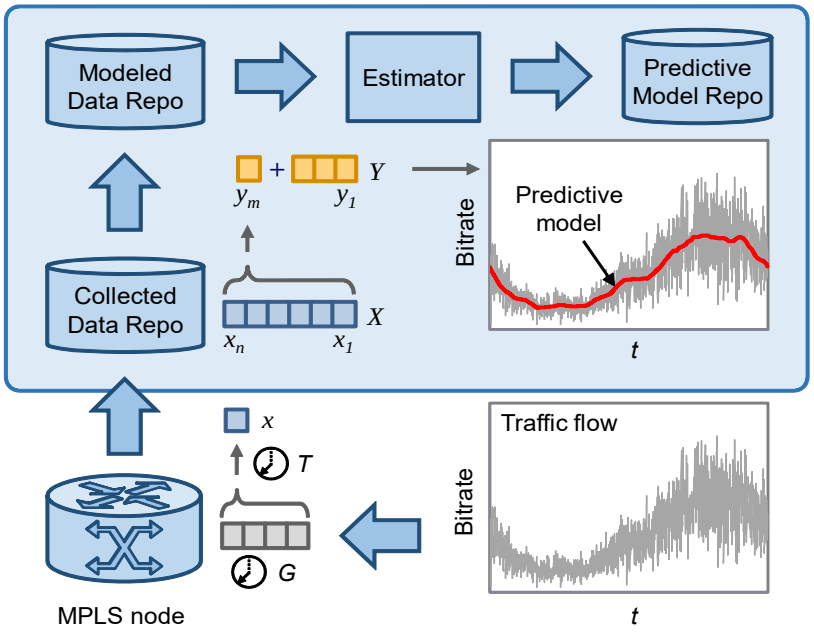

Fig. 5-1 Data analytics modules for traffic flow modelling.

5.1.2 Traffic model estimation

Let us now analyse in more detail the process of data modelling and model estimation that takes place in the previous modules. Fig. 5-2 shows one day of

monitoring data from a traffic flow with granularity G=1 minute. Those samples are aggregated at the MPLS node with a monitoring period of T=15 minutes and sent to the collected data repository (time series X in the figure). Every hour, data from series X is processed and added to a modelled data time series Y containing the average bitrate of the last hour. As we can observe in the figure, data aggregation (averaging in this case) decreases granularity, variability and the total storage requirement. It is worth noticing that the bitrate variability can change along time for various reasons, such as the number of service connections being aggregated into the traffic flow or the type of service of each connection.

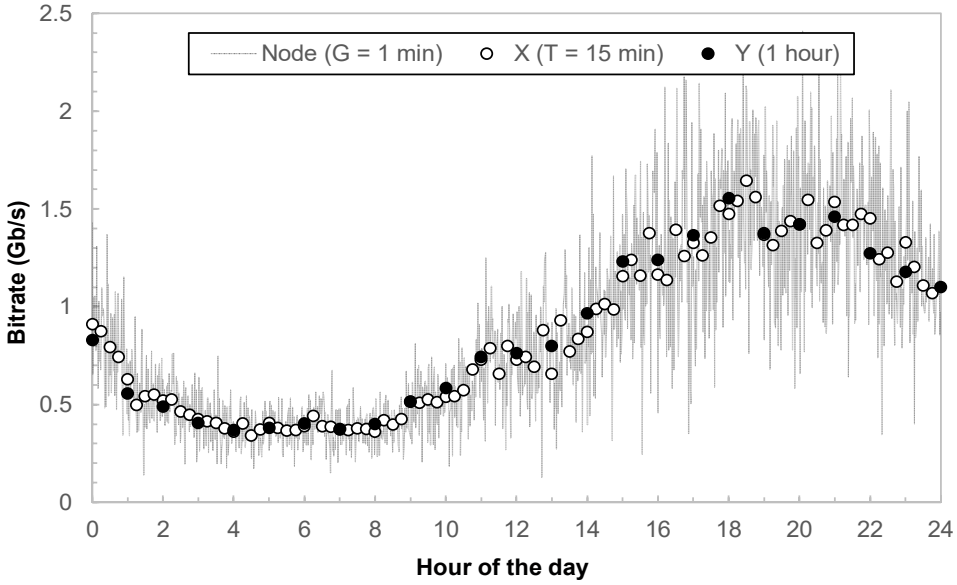

Fig. 5-2 Different aggregation levels for traffic flow time series.

A classical approach to fit the previous time series is to use models from time series analysis, such as the ARIMA model [Wei94]. Such models allow to predict at time t_{n+1} based on k previous values from times $t_{n-i_1},\ldots,t_{n-i_k}$; This entails that, in order to evaluate the model, access to the modelled data repository must be somehow available; let us refer to these models as *time series –valued*. Although time series models have been extensively used in the literature, they present a series of drawbacks when considered for our data analytics framework.

Time series-valued models do not pose a problem when used close the modelled data (e.g., in the network controller). However, they become difficult to evaluate at remote locations where a modelled data repository is not available; one example is when anomaly detection is remotely applied in the network nodes [AV17-1]. Also, input values for these models need to be equally spaced in time to enable their evaluation at any time, thus introducing additional constraints in the modelled data. Last but not least, the discrete nature of the model demands additional processing if a prediction is needed between two discrete times t_n,t_{n+1}. This problem

could be alleviated by increasing the granularity of the data hence obtaining a finer discretisation of the time but at the expense of having more parameters in the model. A more recent and alternative approach is to use machine learning methods such as artificial neural networks (ANN) [Em08], which have been proven to adapt to changing patterns in contrast with more classical time series models.

In light of the previous analysis of time series –valued models we would like to find an alternative approach which overcomes the previous modelling constraints. For instance, it would be desirable to work with predictive models of the form $y=f(t)$ only depending on the absolute time value for a prediction rather than a series of previous modelled data values. These *time point –valued* models can be evaluated regardless of their location. Within this type of predictive models, we aim at those that overcome the discrete nature of time series data by offering a continuous expression for $f(t)$. Some well-known models satisfying this criterion are those based on least-squares (regression) and interpolation. Least-squares offer a wide range of function basis (e.g., polynomials) to accurately approximate the data with relatively few parameters. On the contrary, interpolation allows fitting exactly the data at the expense of having as many parameters as data points to interpolate.

A common problem with both approaches arises when adjusting complex patterns spanning over long time intervals. This usually induces a more complex expression of $f(t)$ along with more parameters to represent it. It usually brings some inefficiencies as well, such as the Runge phenomenon in interpolation. To solve this problem, we can rather choose a continuous, piece-wise model consisting of simpler functions defined over shorter time intervals. For instance, we can consider a piece-wise linear model interpolated over the time series values.

5.2 Time point –valued predictive models

5.2.1 Model estimation

The first model estimation approach aims at obtaining an estimation of the expected mean (or average) bitrate μ and its variance o^2 as a function of time. The proposed estimation algorithm is presented in Table 5-2. It receives a collected time series $Y=X$ for a given traffic flow and the monitoring period T and returns predictive models fitting this data. Specifically, two models (for μ and o^2) consisting of two piece-wise linear functions of a certain number of segments are computed. The main variables used in the algorithm are shown in Table 5-1.

The first part of the algorithm (lines 1-5 in Table 5-2) is a pre-processing phase consisting of grouping traffic values in time series Y by their relative time within the longest identified period. A seasonality detection procedure on the time series [Wei94] is applied to compute the most likely period in the data (line 1). The detection procedure consists in computing the autocorrelation function (ACF) to

detect the distance between consecutive correlation peaks, identified as the duration of the period; *perStart* is set in consequence (e.g., to 00:00h if a daily period is detected).

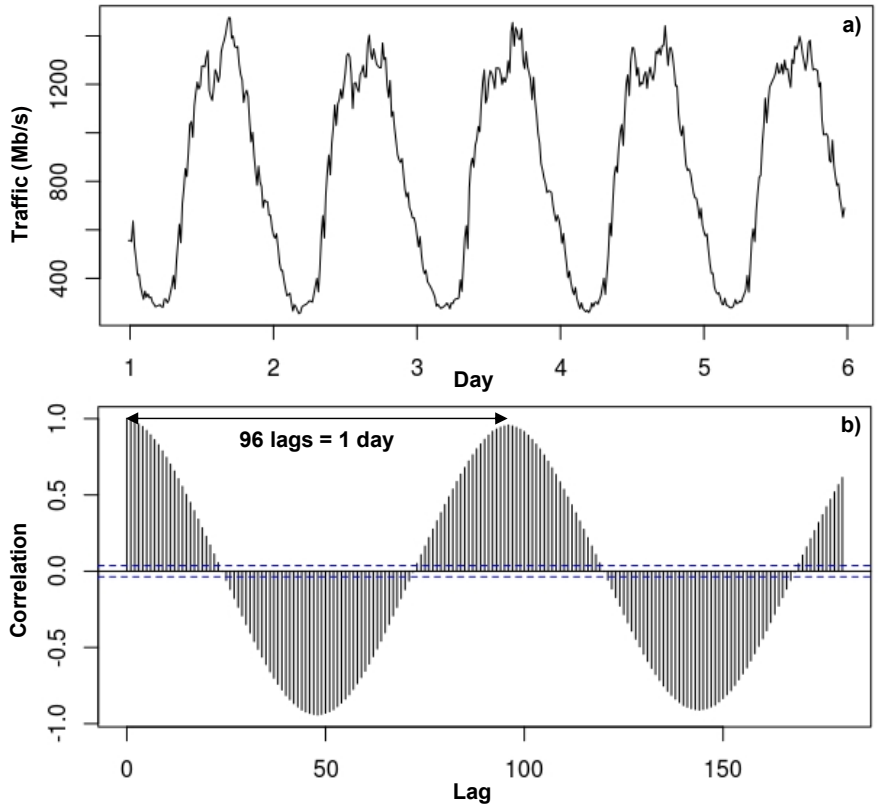

Fig. 5-3 Example of ACF applied to a monitoring time series *Y*.

Fig. 5-3b illustrates the result of the ACF when it is applied to the monitoring traffic series Y from Fig. 5-3a. The bar in the *i*-th lag indicates the correlation that each of the values in the data series has with the value that is exactly *i* positions before. Note that the highest correlation is observed for 96 lags (i.e., one day since samples are collected every 15 minutes). This technique based on the ACF provides a robust periodicity detection whatever the traffic profile is. Since samples are monitored at regular intervals, the number of segments (*nSegm*) of the piece-wise linear functions can be easily computed from *perDur* and *T* (line 2). Once the period has been obtained, data is grouped by segments, i.e., expressed with a time *t* relative to the period. To this aim, every sample in *Y* is retrieved, its relative time *t* computed and the traffic value pushed to the vector that contains all the samples collected at *t*, which is stored in data set *D* (lines 3-5).

Table 5-1: Variable definition for the model estimation.

Name	Definition
perDur	Duration of the period.
perStart	Period starting time.
nSegm	Number of segments of the piece-wise linear functions.
segmLength	Length of each segment.

Table 5-2: Piece-wise linear model estimation

```
   INPUT Y, T
   OUTPUT model
 1:  <perStart, perDur> ← identifyPeriod(Y)
 2:  D ← []; nSegm ← ⌈ perDur / T ⌉
 3:  for y = <time, bitrate> in Y do
 4:     t ← ⌊ ((y.time - perStart) % perDur) / T ⌋
 5:     push(y.bitrate, D{t})
 6:  U ← []; V ← []; μ ← ∅; σ² ← ∅
 7:  segmLength ← perDur / nSegm
 8:  for t in 0..nSegm do
 9:     if t < nSegm then
10:       U[t] ← compute mean u(D{t}) (eq. (5-1))
11:       V[t] ← compute variance v(D{t}) (eq. (5-2))
12:     if t = 0 continue
13:     if t < nSegm then t' ← t - 1 else t ← 0; t'← nSegm - 1
14:     a_μ ← U[t]; a_σ2 ← V[t]
15:     b_μ ← compute slope b(U[t'], U[t], segmLength) (eq. (5-3))
16:     b_σ2 ← compute slope b(V[t'], V[t], segmLength) (eq. (5-3))
17:     μ[t] ← <a_μ, b_μ>; σ²[t] ← <a_σ2, b_σ2>
18:  return model = <perStart, perDur, nSegm, μ, σ²>
```

After the pre-processing phase, D contains the same data in Y but properly grouped to easily compute the coefficients of the linear equation of each segment of the piece-wise linear functions. Every vector $D\{t\}$ is used to compute two consecutive linear equations: for segment $[t\text{-}1, t]$ it is used as ending edge whereas for segment $[t, t+1]$ it is used as starting edge. Thus, for each edge of a segment, the empirical mean and variance are computed according to those typical moment estimators [Ho13] detailed in eq. (5-1) and (5-2) (lines 6-11).

$$u(D) = \frac{1}{|D|} \sum_{i=0}^{|D|-1} D[i] \tag{5-1}$$

$$v(D) = \frac{\sum_{i=0}^{|D|-1} (D[i] - u(D))^2}{|D| - 1} \quad (5\text{-}2)$$

As soon as the empirical mean and variance are available for both edges of a segment, the linear equations (intercept and slope) of that segment for both μ and σ^2 models are computed (lines 12-17). Segments are identified by an intercept a equal to the empirical value at the starting of the segment and a slope b computed from the values at both edges and the segment length, according to equation (5-3). Finally, the model consisting in the obtained period, the number of segments, and the piece-wise linear functions μ and σ^2, is returned (line 18). In practice, *nSegm* is implicitly stored as the size of μ and σ^2.

$$b(d_i, d_j, l) = \frac{d_j - d_i}{l} \quad (5\text{-}3)$$

5.2.2 Model evaluation

Once traffic models are available, they can be used to predict future traffic. Table 5-3 shows the evaluation algorithm that receives a predictive model and the absolute time for which a prediction is needed. First, the absolute time needs to be transformed to a time (t) within the model's period (lines 1-2). Next, the linear equations (slope and intercept) for μ and σ^2 that enclose t are selected (lines 3-6) and finally evaluated (lines 7-8). The algorithm returns the average bitrate and the variance prediction for the requested time (line 9). The number of operations that the algorithm executes is constant and does not depend of the size of the input; therefore, its time complexity is constant, which translates in fast evaluation times in practice. In addition, it requires a constant amount of additional memory apart from that used to load the model (space complexity is linear). This facilitates its utilisation in computational resource scarcity environments. The storage requirements of these models are studied in Section 5.4.

The granularity of the previous prediction is implicitly given by the monitoring period T, which might not be fine enough for some algorithms, such as anomaly detection [AV17-1]. This especially affects the σ^2 model, since the variance of bitrate at a granularity smaller than T (e.g., G) tends to be higher. For illustrative purposes, let us imagine that μ and σ^2 models have been obtained setting a granularity G=1 min and a monitoring period T=15 min. As a result of the implicit traffic aggregation induced by T, we will obtain: *i*) a μ model estimation close to the mean traffic observed at a granularity G and *ii*) a σ^2 model estimation significantly smaller than that observed at a granularity G. Thus, if we want to use the σ^2 model to predict traffic with granularity 1 minute its accuracy needs to be improved. To that end, we propose to use approximated predictions applying corrections derived from the theory of estimation in statistics [Ho13].

Table 5-3: Piece-wise linear model evaluation

```
INPUT model, time
OUTPUT <μ, σ²>
1:  segmLength ← model.perDur / model.nSegm
2:  t ← (time − model.perStart) % model.perDur
3:  s ← ⌊ t / segmLength ⌋
4:  offset ← t % segmLength
5:  [a_μ, b_μ] ← model.μ[s]
6:  [a_σ2, b_σ2] ← model.σ²[s]
7:  μ ← a_μ, + b_μ * offset
8:  σ² ← a_σ2 + b_σ2 * offset
9:  return <μ, σ²>
```

5.3 Time series –valued predictive models

5.3.1 Model estimation

The second model estimation approach consists of fitting ANN models, selected because of its inherent capability of adapting to traffic changes in a non-supervised manner. We consider different ANNs to predict the bitrate of each traffic flow separately. Each ANN receives as input p previous modelled data values of the corresponding traffic flow from the modelled data repository and returns the expected modelled variable value at time t (e.g., the maximum or the average bitrate).

Since the size of an ANN depends on the number of inputs, hidden layers and neurons, we consider ANN models with p inputs, s neurons in a single hidden layer and one output. Consequently, $c=s\cdot(p+1)$ coefficients need to be found to specify every ANN. Aiming at keeping the number of coefficients small, we designed the algorithm in Fig. 5-4 that has to be triggered every time an ANN needs to be estimated. It consists in three phases: *i*) input data pre-processing, *ii*) selection of significant inputs and *iii*) dimensioning of the hidden layer.

In the first phase, a modelled time series Y for the selected traffic flow is retrieved from the modelled data repository. The seasonality detection procedure introduced in the previous section is also applied here to identify the largest period *per* in the time series. This value is used to define the number of inputs of the ANN (i.e., the number of lags in *per*). Based on this, Y is transformed into a dataset D used to fit the ANN model. Every row in D corresponds to a time t within the time series, and every column corresponds to a lag within *perDur*, i.e., $\{y(t\text{-}1),..., y(t\text{-}perDur)\}$

The second phase is an iterative procedure that finds the ANN with the best trade-off between accuracy and number of inputs. This trade-off is captured numerically by the Akaike Information Criterion (AIC) [Em08] (eq. (5-4)); the AIC is computed using the number of parameters n of the model and the residual sum of squares

(RSS) (eq. (5-5)) obtained by evaluating the model against a training data set. Starting with p=*per*, the ANN routine fits an ANN from dataset D and returns the corresponding AIC value. While the AIC value obtained improves the lowest one obtained so far, the best ANN is stored, and p is decremented effectively removing one input. Aiming at reducing the complexity of selecting the input to be removed, we select the lag with lowest ACF. When the minimum AIC is reached, the third phase is executed to increase even more the accuracy of the model by adding hidden neurons until the AIC does not improve. The best ANN is eventually returned, containing the parameters detailed in Table 5-4.

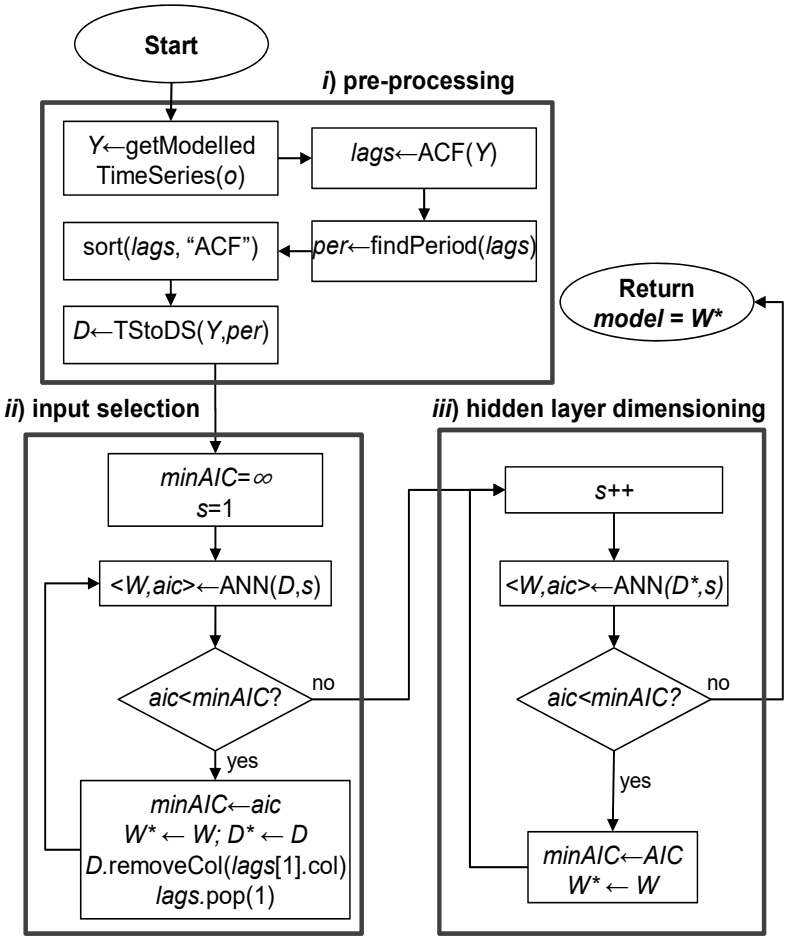

Fig. 5-4 Self-learning ANN fitting algorithm.

$$AIC = 2c + n \cdot \ln(RSS) \tag{5-4}$$

$$RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{5-5}$$

5.3.2 Model evaluation

Once an ANN model is available, it can be evaluated to predict the target modelled variable. The *evaluateANN* algorithm in Table 5-5 can be used to evaluate an ANN model. It receives an ANN model and the absolute time for which a prediction is desired. For the sake of simplicity, let us assume that the modelled data repository stores equally spaced time points and that the input time *for the* algorithm is a multiple of this time separation as well. This constraint is needed since the inputs of the ANN are lags equally spaced in time. Otherwise, the ANN algorithm presented next should be evaluated at the nearest time multiple and then approximate the prediction to the desired input time, for instance, by interpolation.

Table 5-4: Parameters of the ANN model

Name	Definition
P	Set of inputs neurons.
S	Set of neurons of the hidden layer. Each neuron s contains the activation function $f_s(x)$.
inWeights	Real-valued matrix containing the weights of every connection $(p, s) \in P \times S$.
outWeights	Real-valued vector containing the weights between every neuron $s \in S$ and the output.

Initially, the input time value is used to retrieve from the modelled data repository the set of p previous modelled data values needed as input for the ANN (line 1). It might happen, however, that some of the input data were missing if the requested prediction is far ahead in the future and some lags are not yet in the repository. For these missing values, the algorithm is recursively called thus obtaining an estimation for the input data (lines 2-3). It is worth noticing that this recursive approach introduces an accumulated error in the originally requested prediction. Once completed, the input data is propagated through the neural network by properly combining it with the weights of the connected neurons. Formally, this is achieved by running an iterative procedure on the hidden layer. For each $s \in S$, each input value is multiplied with the weight connecting p and s (lines 4-8) and added to a temporary value. Next, the temporary value is evaluated by the activation function of s, $f_s(x)$, weighted and added to the final prediction value y_{pred} (line 9). Once the iterative procedure ends, the final prediction is returned (line 10).

The time complexity of the previous algorithm is $O(s\,p)$, plus the time needed to retrieve the input modelled data from the repository. Therefore, the evaluation algorithm depends on the size of the ANN, in contrast with the piece-wise linear models, which present o(1) time complexity for their evaluation.

Table 5-5: ANN model evaluation

```
INPUT model, time
OUTPUT y_pred
I ← modeledDataRepo.getInputs(model.P, time)
for <y, t> ∈ I do
   if y = ∅ then y ← evaluateANN(model, t)
y_pred ← 0
for s ∈ model.S do
   x ← 0
   for p ∈ model.P do
      x ← x + I[p].y · model.inWeights[p,s]
   y_pred ← y_pred + model.f_s(x) ·model.outWeights[s]
return y_pred
```

5.4 Numerical results

This section starts by presenting the traffic flows used in the numerical evaluation of the two proposed model estimation approaches. Next, both traffic modelling approaches are evaluated by analysing different performance metrics.

5.4.1 Traffic flow generation

Due to the diversity of service types that can be conveyed in metro and core flows, we ran simulations in iONESim to generating monitoring data according to two differentiated traffic profiles. The first traffic profile, named as *Users*, represents the traffic aggregation of end users consuming high-bandwidth applications such as video-on-demand or live TV, with higher activity at evening hours, i.e. prime-time [Ve14.3]. The second traffic profile, named as *Datacenter* (DC), aggregates traffic of DC to DC connectivity services required for dynamic management activities of distributed DCs, such as DB synchronisation or virtual machine (VM) migration [Be92]. The daily expected bitrate of these two profiles is illustrated in Fig. 5-5.

The monitoring time series X for each traffic profile is generated using the traffic generation framework presented in [AV16] and included as part of iONESim. Based on this framework, a time series is generated by combining a deterministic average $f(t)$ (as in Fig. 5-5) with a stochastic component. To do so, a random variable ε representing random variable traffic is defined assuming a 0-centred normal probability distribution. Therefore, monitoring samples for every traffic flow are generated applying eq. (5-6). Aiming at correlating burstiness with peak hours, a time dependency is introduced in the standard deviation of ε as stated by eq. (5-7) assuming a constant *coefficient of variation* (*CV*) in the traffic [Ho13]. Based on this procedure, training and validation time series with several months of monitored bitrate were generated at a granularity G=1 minute for the Users and

DC traffic profiles generated with iONESim. Fig. 5-6 and Fig. 5-7 illustrate two days of this data for each of the traffic profiles. The traffic generation approach presented here will be used in the forthcoming chapters.

$$X(t) = f(t) + \varepsilon \tag{5-6}$$

$$\sigma_{\varepsilon}(t) = \frac{f(t)}{CV} \tag{5-7}$$

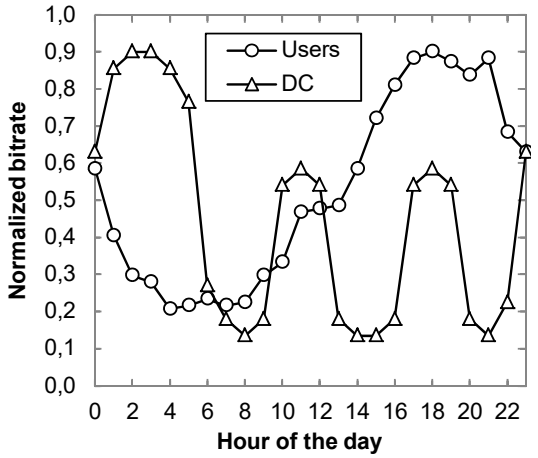

Fig. 5-5 Average daily bitrate of Users and Datacenter profiles.

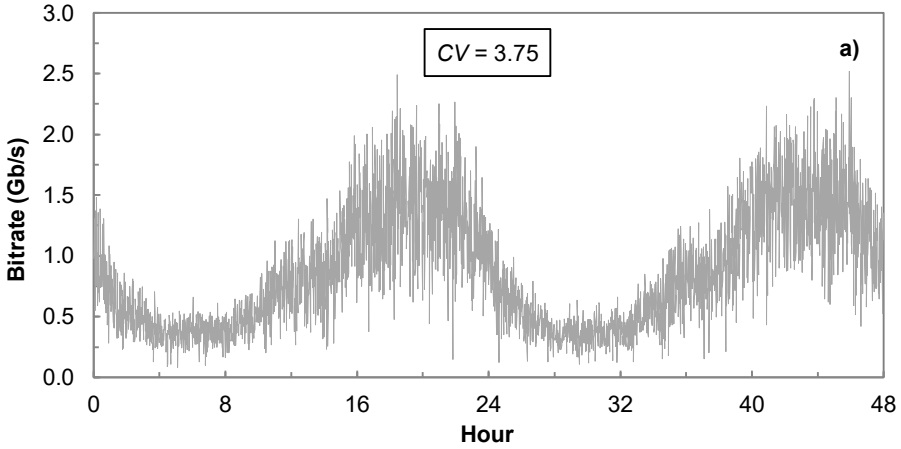

Fig. 5-6 Two days of monitoring data of Users traffic generated using iONESim.

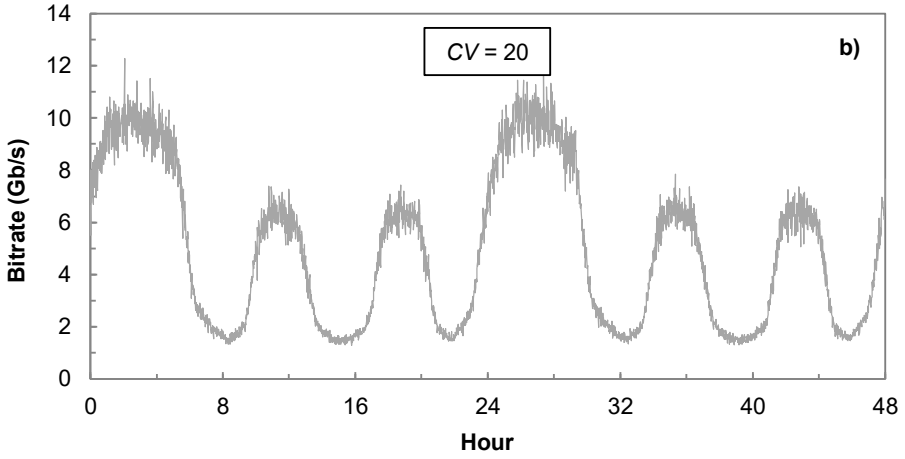

Fig. 5-7 Two days of monitoring data of DC traffic generated using iONESim.

5.4.2 Piece-wise linear model estimation

Let us start evaluating the piece-wise linear models by first focusing on analysing the quality of μ model estimation under different monitoring periods T of 1, 5, 15, 30 and 60 minutes. For each T, we retrieved time series X from the training data set containing several months of monitoring data used as input to the estimator algorithm in Table 5-2 (we assume no modelling of the data, i.e., Y=X). To check the quality of the estimation, we compared it against a new, validation time series Y^* for each traffic profile; in particular, for each data value $y \in Y^*$ the error of the prediction was computed as the relative difference between the prediction and the real monitored value y. The average and maximum error were then compared for different values of T. By observing the average error we find values under 1% for both traffic profiles and for T lower than 30 min. However, when looking at the maximum error (i.e., the worst prediction) we notice important differences between both traffic profiles. While the Users profile yields maximum error values below 2% for all T, those for the DC profile remain low only when $T \leq 15$ min, while exceeding 10% for larger T; this is caused by the combination of abrupt changes in its daily profile and the loss of information because of aggregation.

Consequently, setting T = 15 min provides a good tradeoff between information loss and prediction quality and hence, we fix this value for the rest of this study. Similar experiments were conducted to evaluate the estimation of the variance σ^2. Fig. 5-8 shows the error resulting from comparing the estimation of σ^2 against the variance used to generate the training time series. It can be observed that the estimation offers a 0-centred error bounded by ±10%. Results are shown averaged for the Users and DC profiles, yielding similar results for both individually.

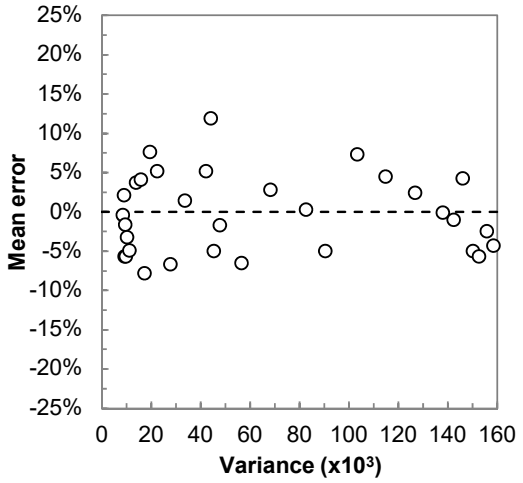

Fig. 5-8 Mean error of σ^2 estimation vs days of monitoring.

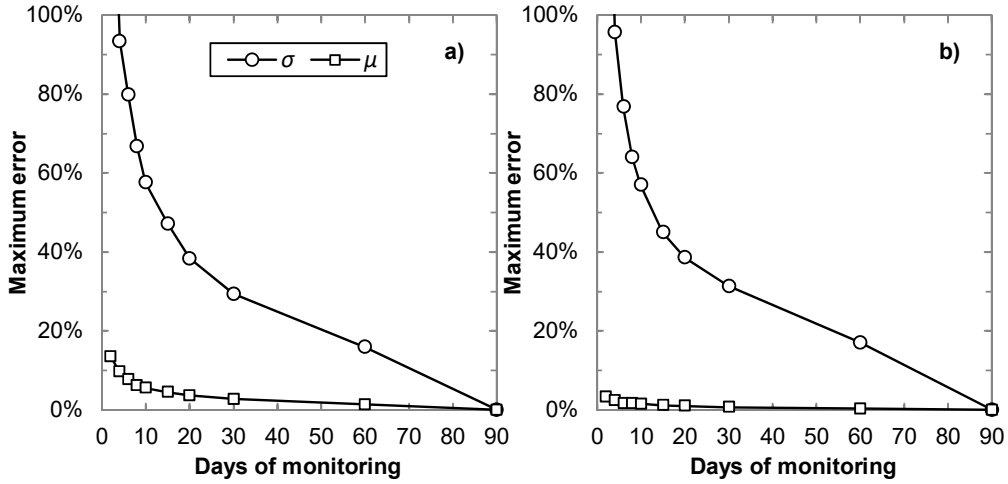

Fig. 5-9 Maximum error of σ^2 and μ vs days of monitoring for the Users (a) and the DC (b) traffic profiles.

Let us now analyse the impact of the amount of training data (i.e., $|Y|$) in the quality of the predictive models. Monitoring traffic during the right period is crucial to producing quality models while minimising the time for new model availability. To evaluate this, we conducted experiments where μ and σ^2 are estimated and evaluated varying $|Y|$ between 2 days and 3 months.

Fig. 5-9a-b show the maximum error for σ^2 and μ estimations for different values of $|Y|$. Values are normalized to the ones obtained with $|Y|$=3 months, which offered an acceptable error. Although μ can be estimated with less than 5% maximum error in about 10 days, a maximum error of 60% is obtained for σ^2 for the same time. To decrease the maximum error $|Y|$ need to be increased up to 2 months to keep maximum prediction errors under 20%. Henceforth, we consider

$|Y|$=3 months of traffic monitoring to train models to accurately fit the behaviour of metro-flows. Fig. 5-10 shows one day of monitoring traffic data, as well as the prediction of the μ model (red line) and the confidence interval at 95% that is obtained using the σ^2 model (dashed lines).

Finally, let us analyse the storage requirements for the piece-wise linear models. Each traffic model requires $2 \cdot (2N+1)$ floating point numbers to be stored, where N is the number of model pieces. As an example, in a 100-node network, each node would require 153 KB to store all the traffic flow models originated in the node to every other node with daily periodicity, being this value increased to 4 MB if monthly periodicity is used.

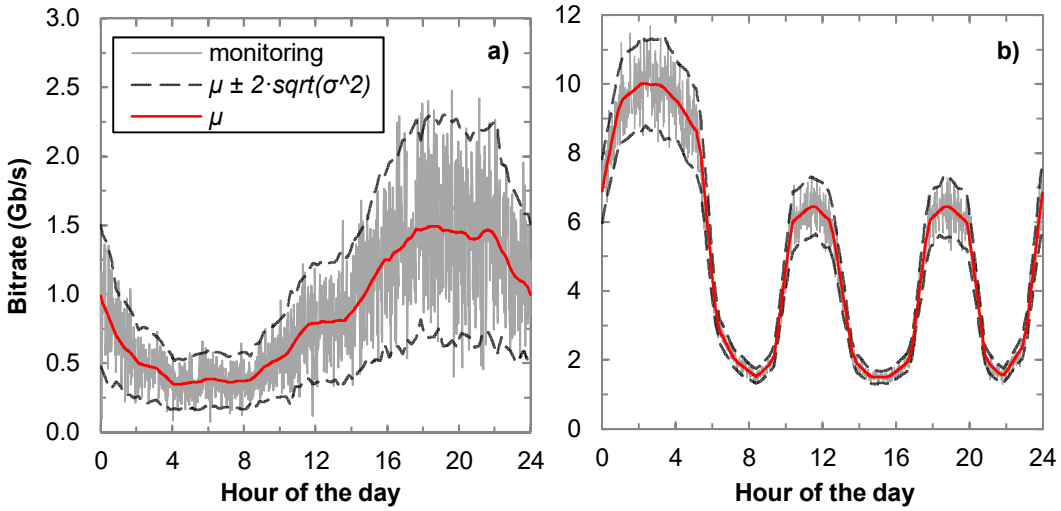

Fig. 5-10 Prediction of min/max/avg for Users (a) and DC (b) traffic profiles.

5.4.3 ANN-based model estimation

To evaluate the ANN-based model estimation approach, we retrieved time series X from the training data set containing several months of monitoring data. For each time series X, a modelled time series Y was computed to obtain the maximum hourly bitrate. In this case, we chose to model the maximum hourly bitrate aiming at obtaining ANN-based models capable of predicting the peak bitrate value along one hour, used for network design and reconfiguration. Based on this data, ANN models were trained applying the fitting algorithm in Fig. 5-4.

Results shown in Fig. 5-11b illustrate the average input size p of ANN models obtained in the input selection phase. Recall that during the input selection phase, the number of inputs p is decreased aiming at minimising the AIC value. We observe that the minimum AIC is on average reached at p=4, being mainly selected those inputs from t-1 to t-4. Results from the hidden layer dimensioning phase fixing p=4 are shown in Table 5-6, for a number of hidden neurons ranging from 1

to 3. Note that the minimum AIC is obtained for *s*=2, which results in an ANN model with 10 coefficients that accurately predicts the output variable with a good trade-off between average and maximum relative errors (2.64% and 9.55%, respectively).

Finally, let us analyse the degree of adaptation of ANN models under evolutionary traffic scenarios. To that end, we generated evolutionary time series according to two different scenarios. In the first scenario (*smooth*), the bitrate experiments a transformation from the current traffic pattern to an unexpected one in a time span of two weeks, whereas in the second scenario (*sharp*) the bitrate suffers a multiplicative increment on its bitrate values in a time span of one month, although its normalized traffic pattern remains constant.

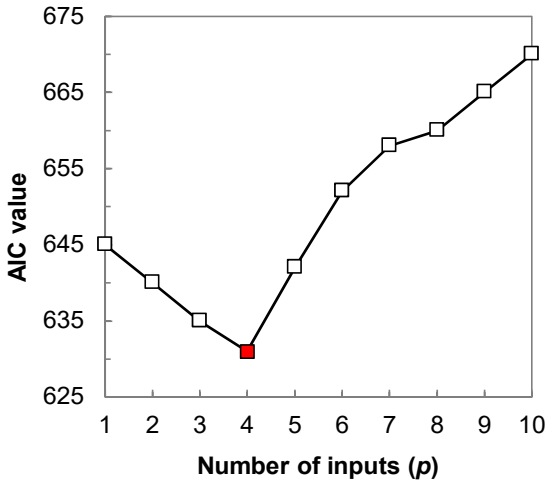

Fig. 5-11 AIC values for the ANN input selection phase.

Table 5-6 ANN hidden layer dimensioning phase

s	AIC	Error (%)	
		Avg.	Max.
1	631.43	2.07%	10.44%
2	616.65	2.64%	9.55%
3	644.46	3.21%	10.68%

Fig. 5-12 shows the adaptation of a sample ANN model to the two types of evolutionary traffic, without performing any refitting in the model. By looking at the smooth evolutionary traffic scenario in Fig. 5-12a we can see how the ANN model is capable of adapting from the old (red) to the new (blue) traffic pattern while keeping its original predictive accuracy without refitting. This is an

important feature of ANN-based estimation in contrast with piece-wise linear models, since the former provides no means of adaptation in front of smoothly changing traffic. In contrast, the sharp evolutionary scenario in Fig. 5-12b reflects the impossibility of the ANN model to adapt to abrupt scaling in the traffic pattern thus requiring re-running the ANN estimation algorithm.

Let us finish de study by analysing the storage requirements for the ANN models. Assuming ANN models with p=4 inputs and s=2 neurons in the hidden layer, it would require 10 floating-point numbers to store each model. As an example, in a 100-node network, each node would require 3.96 KB to store all the traffic flow models originated in the node to every other node with daily periodicity (38 times less storage than piece-wise linear models). However, in practice such large variations in traffic are very unlikely in the short term (except for DDoS attacks [Lau00]). In this regard, it is worth mentioning the role of incremental planning to detect long-term trends in contrast with in-operation network planning, based on parameters like the compound annual growth rate (CAGR) [Ko13].

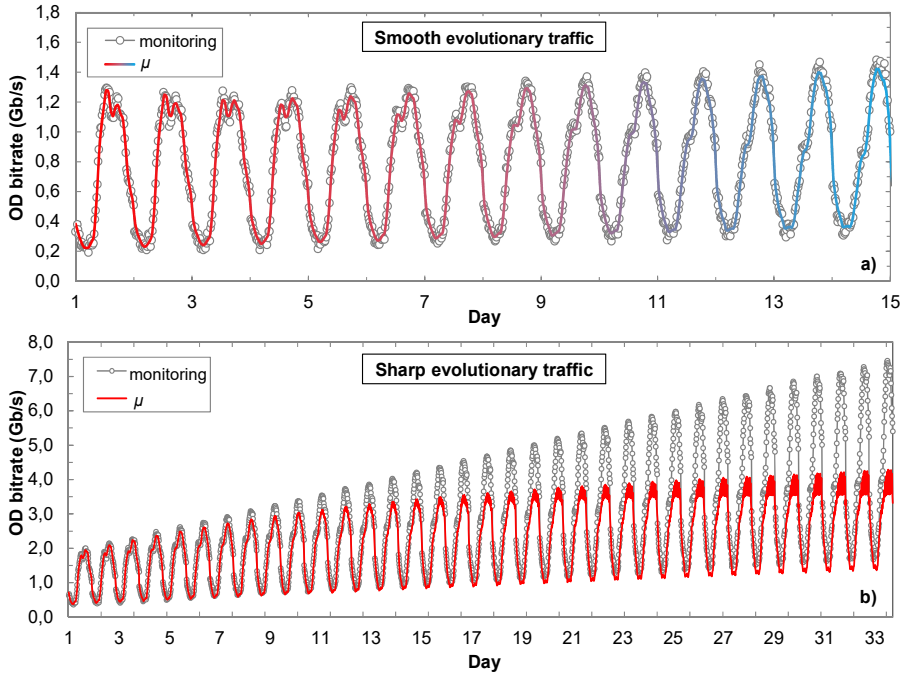

Fig. 5-12 ANN adaptation to smooth (a) and sharp (b) evolutionary bitrate.

5.5 Conclusions

In this chapter, a centralised data analytics framework has been presented to enable network controllers to obtain predictive traffic flow models based on monitoring data. Monitoring data is collected from MPLS routers at regular

intervals and sent to a centralised repository where a time series for every monitored flow is stored. Periodically, modelled data is produced, conveniently summarised the collected data and using it as the input of an estimator algorithm.

Two different approaches have been proposed to produce traffic models, namely a Piece-wise linear estimation algorithm and an ANN-based estimation algorithm, both capable of transforming monitoring data into quality traffic prediction. Both model estimation approaches were numerically evaluated against monitoring time series generated in iONESim.

Although the numerical evaluation shows comparable predictive quality for the two proposed approaches, we have identified a series of opposed trade-offs between the two approaches, relative to either their theoretical properties or from their numerical evaluation. These trade-offs are summarized in Table 5-7.

Table 5-7 Statistics vs ANN –based estimation

	Piece-wise linear model	**ANN model**
Pros	Evaluation takes o(1) and only depends on time.	Adapts to evolutionary traffic. No assumptions about underlying traffic distribution.
Cons	Does not adapt to evolutionary traffic. Assumes normality in the traffic.	Evaluation takes O($s\,p$) and depends on modelled data. Predicts a single value.

In the following chapter, we propose to apply some of these traffic modelling approaches to predict the OD bitrate in a core VNT. By combining this prediction with optimisation techniques, the VNT can be periodically adapted to current and future traffic.

Chapter 6

VNT reconfiguration based on traffic prediction

In this chapter, we target at adapting the VNT to future traffic conditions. To that end, predictive traffic models for the OD traffic can be used as input of the VNT reconfiguration problem based on traffic prediction (VENTURE). The VENTURE problem is formally stated and formulated as a MILP model. Given its complexity, a heuristic algorithm providing a better trade-off between optimality and complexity is designed. The discussion is finally supported by results from exhaustive simulation over a realistic scenario.

6.1 VNT design and reconfiguration options

Several approaches to design and dynamically reconfigure the VNT can be devised. In this section, we start reviewing two of them: *i*) the static VNT design and *ii*) the threshold-based VNT capacity reconfiguration. For the rest of this chapter, let us assume an underlying optical layer with unlimited spectrum availability.

In the static VNT design, the topology is designed and dimensioned to cope with the maximum of daily traffic forecast for every OD pair during a planning period. The resulting topology is thus, capable of supporting the traffic at any time during that period provided a perfect traffic forecast. Fig. 6-1 presents an example for a seven-node VNT, where the capacity of every vlink supports the maximum daily traffic volume. For illustrative purposes, the plot in Fig. 6-1a shows the variability in link 6-1 that needs to be dimensioned with a capacity of 200 Gb/s. Fig. 6-1b shows the resulting VNT with the capacity of every vlink. It is clear, in view of Fig.

6-1a, that the main drawback of the static VNT design is over-provisioning since most of the available capacity in the VNT will remain underutilised along the day.

Aiming at reducing capacity over-provisioning, that of the vlinks can be adapted over time instead of allocating a constant amount of resources. Let us assume that the capacity of the existing vlinks can be increased and decreased to follow the traffic variations but no new vlinks can be created or removed, keeping hence the VNT invariant. Traffic can be monitored at IP routers, and when the amount of traffic through a vlink reaches some threshold (e.g., 90%) the network controller can increase the capacity of such vlink by setting-up a parallel lightpath between the two IP routers; conversely, unused capacity can be released by tearing down lightpaths.

The example in Fig. 6-2a presents monitored traffic data captured during the last two hours in node 6, where traffic from that node to every other node in the VNT (labelled as 6→N), from node 6 to node 7 (6→7), and from node 6 to every other node except to node 7 (6→N\{7}) are plotted. Fig. 6-2b shows the initial VNT where every vlink is supported by a 100 Gb/s lightpath in the underlying optical layer; the IP/MPLS path for OD 6-7 is also shown. A 90% threshold is configured and, in the event of threshold violation, the capacity of some vlinks is increased. In our example, two threshold violations for vlinks 1-6 and 1-7 are received, so the VNT capacity is updated (Fig. 6-2c). It is worth noting that IP/MPLS path for OD 6→7 is not affected by the VNT reconfiguration. As shown in the example, the threshold-based reconfiguration can adapt the VNT capacity to traffic changes, so resources in the optical layer are allocated only when vlinks need to increase their capacity. However, the same number of transponders as in the static VNT design approach needs to be installed in the IP routers; for instance, in the example in Fig. 6-2 two transponders are installed in routers 6 and 7 and another four in router 1 reserved for vlinks 1-6 and 1-7.

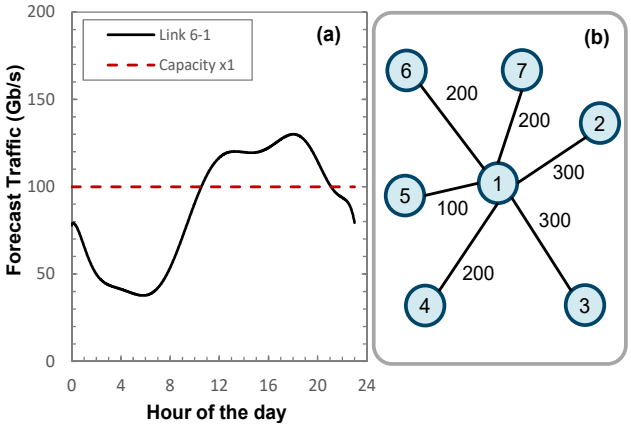

Fig. 6-1 Static VNT design.

Let us assume now that, instead of monitoring vlink capacity usage, OD traffic is monitored in the routers. Indeed, analysing the plots in Fig. 6-2a we realise that traffic 6→7 is responsible for the registered traffic increment. In this case, let us assume that new vlinks can be created/removed in addition to increasing the capacity of the existing ones, so the VNT is changed. We propose an approach where OD traffic is periodically analysed, and the current VNT is reconfigured accordingly. An example following this approach is illustrated in Fig. 6-3, where the OD traffic 6→7 is analyzed at t=60, and a maximum value (e.g., 90 Gb/s) is predicted for the next hour. Then, a new vlink between nodes 6 and 7 can be created by establishing a lightpath on the optical layer and traffic 6→7 rerouted (Fig. 3b). Note that this solution reduces two transponders to be installed in router 1 compared to the previous approaches. It is clear that this reduction will happen when the amount of traffic is large enough. In particular, when the amount of traffic exceeds the capacity of the installed transponders (e.g., 100 Gb/s) direct vlinks can be created for part of that traffic, while the residual part could be routed through a different IP/MPLS path.

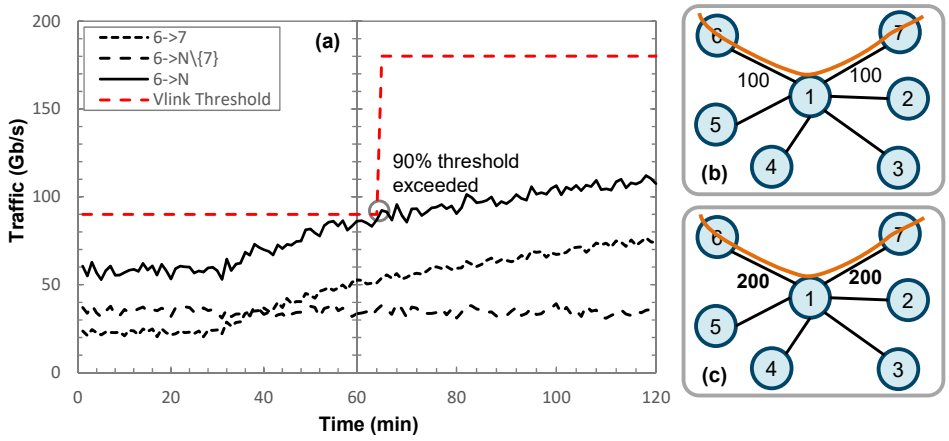

Fig. 6-2 Threshold-based VNT capacity reconfiguration.

In order to adapt the VNT to changes in the traffic, we propose a predictive model built upon the monitored OD traffic data. For every OD pair, meaningful statistical values are predicted (e.g., the maximum bitrate for the next hour) and used to adapt the VNT to meet the future traffic matrix, assuming that every OD traffic can be conveyed through two IP/MPLS paths at the most. We call this approach as VENTURE.

Let us present the proposed modules for the VENTURE approach, extending those presented in Chapter 5 for traffic modelling. Let us assume that traffic monitoring data is collected at the edge IP routers at regular intervals, e.g., every 15 minutes. Every edge router collects a set of samples for the traffic to every other destination router, which is stored in a collected data repository (Fig. 6-4). Note that since we focus on OD traffic monitoring, $|N| \cdot (|N|-1)$ traffic samples need to be stored at every monitoring interval, where $|N|$ is the number of routers.

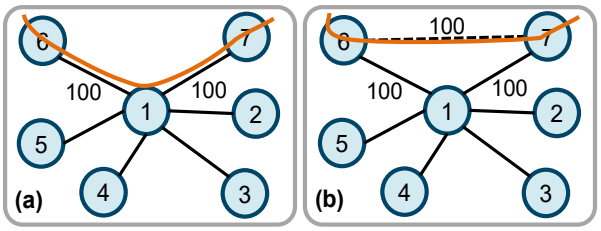

Fig. 6-3 VNT reconfiguration based on OD traffic prediction.

Following a predefined period, e.g., every hour, a time series from the collected data repository is retrieved for each OD pair and pre-processed applying data stream mining sketches to conveniently summarise collected data thus producing modelled data representing the OD pair that is stored in a modelled data repository. Modeled data includes, among others, for every OD the minimum, maximum, average, and last collected bitrate within the hour.

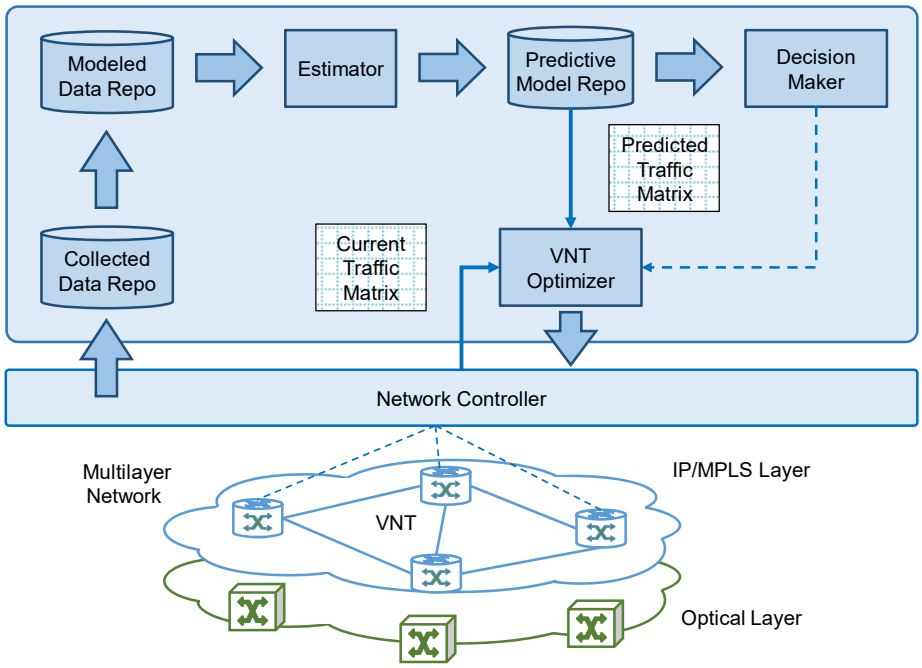

Fig. 6-4 Applying the OAA loop for VNT reconfiguration.

The set of modelled variables for the current period t is stored in a repository together with variables belonging to previous periods. A prediction module based on machine learning techniques generates the OD traffic matrix predicted for the next period that is used by a decision maker module to decide whether the current VNT needs to be reconfigured. In case that a reconfiguration needs to be performed, the decision maker module triggers the execution of a VNT reconfiguration algorithm in a VNT optimiser module receiving as input the

current and the predicted OD traffic matrices. Once the VNT optimiser returns a solution, the network controller would be responsible for implementing the changes in the data plane.

6.2 The VENTURE problem

Next, we first formally state the VENTURE problem to be solved in the VNT optimiser and devise a MILP to model it. In light of the complexity of the problem, a heuristic algorithm is eventually devised.

6.2.1 Problem statement

The VENTURE problem can be formally stated as follows:

Given:

- The current VNT represented by a graph G(N, E'), being N the set of routers and $E' \subseteq E$ the set of current vlinks. Set E is the set of all possible vlinks connecting two routers,
- the set P with the transponders available in the routers; every transponder with capacity B,
- the current traffic matrix D,
- the predicted traffic matrix OD. The bitrate b_o of OD pair o must be served following one single path. Only in the case that b_o is enough to fill transponders with an amount over a given boundary usage tbu, the bitrate of pair o can be split into two flows and served through different paths.

Output: The reconfigured VNT $G^*(N, E^*)$, where $E^* \subseteq E$, and the paths for the traffic on G^*.

Objective: Maximize current and predicted served traffic matrices, while minimising the total number of transponders used.

6.2.2 Mathematical model

Note from the problem statement that both, the current and the predicted traffic matrices must be served. Consequently, we generate an input traffic matrix OD, where every pair o is the maximum of both, the current and the predicted traffic. In addition, a parameter k_o will be used to specify whether pair o can be served using one or two paths.

The following sets and parameters are defined:

Topology:

N	set of routers, index n.
E	set of all possible vlinks, index e.
$E^+(n)$	subset of E with vlinks outgoing from router n.
$E^-(n)$	subset of E with vlinks incident in router n.

Traffic:

OD	set of origin-destination pairs, index o. Every o is defined by the tuple $<s_o, t_o, d_o, b_o>$, where s_o and t_o specify the source and target nodes, d_o the currently served bitrate and b_o the maximum of current and predicted bitrate to serve for pair o, respectively.
k_o	maximum number of paths to serve pair o; $k_o = 2$ if $b_o \geq tbu$; $k_o = 1$ otherwise.

Equipment:

P	set of transponders, index p. Every transponder consists of one transmitter (tx) and one receiver (rx).
$P^+(n)$	subset of tx transponders in router n.
$P^-(n)$	subset of rx transponders in router n.
$P(n)$	subset of transponders in n. $P(n) = P^+(n) \cup P^-(n)$.
B	capacity of every transponder.

The decision variables are:

x_p	binary, 1 if transponder p is used, 0 otherwise.
x_{pe}	binary, 1 if transponder p is used to support vlink e, 0 otherwise.
x_{ok}	integer, fraction of bitrate of pair o served through path k.
x_{oke}	integer, fraction of bitrate of pair o served through path k using vlink e.
z_{oke}	binary, 1 if pair o is routed using path k through vlink e, 0 otherwise.
y_n	integer+, number of transponders used at router n.
v_o	integer+, fraction of unserved bitrate of pair o.

Then, the proposed ILP formulation is as follows:

$$\text{(VENTURE)} \quad \min \quad (|P|+1)\cdot\sum_{o\in OD} v_o + \sum_{n\in N} y_n \tag{6-1}$$

subject to:

$$\sum_{e\in E^+(n)} z_{oke} - \sum_{e\in E^-(n)} z_{oke} = \begin{cases} 1 & \forall o \in OD, k = 1..k_o, n = s_o \\ 0 & \forall o \in OD, k = 1..k_o, \\ & n \in N \setminus \{s_o, t_o\} \\ -1 & \forall o \in OD, k = 1..k_o, n = t_o \end{cases} \tag{6-2}$$

$$x_{oke} \leq b_o \cdot z_{oke} \quad \forall o \in OD, k = 1..k_o, e \in E \tag{6-3}$$

$$x_{oke} \leq x_{ok} \quad \forall o \in OD, k = 1..k_o, e \in E \tag{6-4}$$

$$x_{ok} - b_o \cdot (1 - z_{oke}) \le x_{oke} \quad \forall o \in OD, k = 1..k_o, e \in E \tag{6-5}$$

$$\sum_{k=1}^{k_o} x_{ok} + v_o \ge b_o \quad \forall o \in OD \tag{6-6}$$

$$\sum_{k=1}^{k_o} x_{ok} \ge d_o \quad \forall o \in OD \tag{6-7}$$

$$\sum_{o \in OD} \sum_{k=1}^{k_o} x_{oke} \le B \cdot \sum_{p \in P^+(i)} x_{pe} \quad \forall e = (i,j) \in E \mid i,j \in N \tag{6-8}$$

$$\sum_{o \in OD} \sum_{k=1}^{k_o} x_{oke} \le B \cdot \sum_{p \in P^-(j)} x_{pe} \quad \forall e = (i,j) \in E \mid i,j \in N \tag{6-9}$$

$$\sum_{e \in E^+(n)} x_{pe} \le x_p \quad \forall n \in N, p \in P^+(n) \tag{6-10}$$

$$\sum_{e \in E^-(n)} x_{pe} \le x_p \quad \forall n \in N, p \in P^-(n) \tag{6-11}$$

$$\sum_{p \in P^+(n)} x_p \le y_n \quad \forall n \in N \tag{6-12}$$

$$\sum_{p \in P^-(n)} x_p \le y_n \quad \forall n \in N \tag{6-13}$$

The multi-objective cost function (6-1) minimises both, unserved traffic and used transponders, where the highest cost corresponds to the first term. This is achieved by considering one cost unit of unserved traffic equal to equipping all the available ports in the network.

The network flow constraints in (6-2) define paths on the topology for every OD pair. Each of these paths has a continuous capacity assigned along its route, as imposed by constraints (6-3)-(6-5): constraint (6-3) allows setting a capacity in each link of a path whereas constraints (6-4) and (6-5) force this capacity to remain constant along all traversed link. Note that optimal solutions might include loops that they can be safely removed in a post-processing phase.

Although constraint (6-6) allows serving only a fraction of the total capacity b_o of every OD pair; it has to include at least the currently served bitrate as stated in constraint (6-7). This constraint ensures that the OD traffic matrix before the reconfiguration remains served in any solution.

Constraints (6-8)-(6-13) deal with transponder equipment. Constraints (6-8) and (6-9) assign transmission and reception transponders to vlinks, respectively to support the capacitated paths. It is worth noticing that two transponders need to

be equipped to support one capacity increment for a vlink: a transmission transponder in the vlink's source node (constraint (6-8)) and a reception transponder in the vlink's target node (constraint (6-9)). Constraints (6-10) and (6-11) prevent from equipping the same transponder to increase the capacity of different vlinks. Finally, constraints (6-12) and (6-13) compute the number of transponders to be installed in every router, represented by the maximum between the number of transponders used for transmission and reception.

The size of the proposed formulation is $O(|N|^4+|P|\cdot|N|^2)$ variables and $O(|N|^4+|P|\cdot|N|)$ constraints. As an example, the size of the above formulation for the network instance with k_o=2 and 14 nodes presented in section 0 is of $2\cdot10^5$ variables and 10^5 constraints. As a result, solving the proposed formulation becomes impractical for realistic scenarios even using commercial solvers; in our tests, solving times were longer than 10h. Consequently, we developed a heuristic algorithm that provides a much better trade-off between optimality and complexity.

6.3 Basic heuristic algorithms

The heuristic algorithm devised to solve the VENTURE problem is based in the *greedy* notion that OD pairs with higher bitrate should be routed first to obtain cost-saving solutions, as illustrated in Fig. 6-5: starting from an empty VNT, Fig. 6-5a shows an example solution where OD 1→3 is routed first through direct vlink 1→3. Next, OD pairs 1→2 and 1→2 are routed through their respective direct vlinks to minimize transponder utilization locally. As a result, 3×100 Gb/s transponders are needed. In contrast, Fig. 6-5b illustrates the same scenario when OD pairs are processed from higher to lower OD bitrate, only needing in this case 2 transponders to allocate the same capacity.

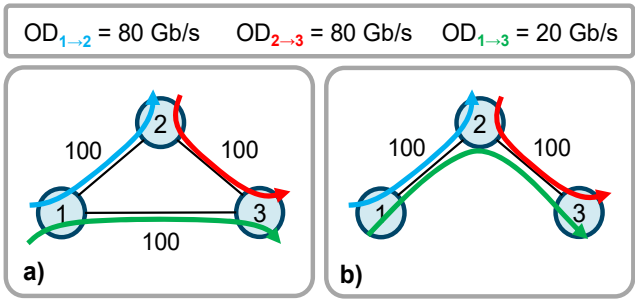

Fig. 6-5 Rationale behind the heuristic algorithm.

Based on this idea, we divide the input traffic matrix into two subsets for their processing: the first subset is formed by those OD pairs with enough capacity to fill transponders above a pre-defined threshold, whereas the second subset contains all the remaining OD pairs of less capacity. The heuristic algorithm is then separated

into three ordered stages, where high OD pair capacity is allocated first through a direct vlink in the hope of minimising the cost of the resulting VNT.

Table 6-1 Main Algorithm

```
   INPUT G(N,E'), D, OD, B, tbu
   OUTPUT G*, F
 1:  Q ← ∅, U ← ∅
 2:  for each d ∈ D do dealloc(G, d)
 3:  for each e ∈ E' do
 4:     setCapacity(e, 0)
 5:     releaseTransponders(e)
 6:  for each o ∈ OD do
 7:     Q ← Q ∪ {<o, g_o, l_o> = splitOD(o, B, tbu)}
 8:  <Q, F'> ← PhaseI(G, Q)
 9:  <G', Q, F''> ← PhaseII(G, Q)
10:  for each q ∈ Q do
11:     U ← U ∪ {<o, u_o = g_o + l_o>}
12:  if U=∅ then return <G', F' U F''>
13:  <G*, F'''> ← PhaseIII(G', U, thr)
14:  if F'''=∅ then return INFEASIBLE
15:  return <G*, F' U F'' U F'''>
```

The heuristic algorithm to solve the VENTURE problem is shown in Table 6-1. After deallocating current traffic and releasing used resources (lines 2-5 in Table 6-1), bitrate b_o of OD pairs is split into two different flows and stored in set Q: flow g_o carries bitrate enough to fill transponders with an amount over tbu and flow l_o carries the remaining bitrate; these flows will be routed through different paths (lines 6-7). Next, the first two phases focus on routing every flow g_o through the direct vlink connecting source and destination routers (lines 8-9); sets F stores the path of the flows. The first phase selects those flows for which a direct vlink already exists in the current VNT and the second phase does the same for the rest of flows thus, creating new direct vlinks. After these two phases, the residual bitrate u_o is checked and stored in set U. If all traffic has been already served, the algorithm ends (lines 10-12); otherwise, OD pairs are sorted by the amount of unserved bitrate and the third phase eventually routes the unserved bitrate by possibly increasing the capacity of existing vlinks or by adding new ones (line 13). The reconfigured VNT and the new routing are eventually returned.

The algorithm for the first phase is detailed in Table 6-2. The original uncapacitated topology is used to route flows g_o through an existing direct vlink to introduce inertia to the changes in the current topology (line 3 in Table 6-2). The number of transponders to be allocated in the end routers of the direct vlink is computed as the minimum between the amounts of transponders needed to allocate g_o and the unused transponders (lines 4-7); those transponders are allocated to add capacity to the direct vlink (line 8) and a shortest path is computed on the resulting VNT (line 9). In case that a path is found (i.e., capacity was added to the direct vlink), the path is allocated and the amount of served bitrate reduced from the one

requested (lines 10-13). The updated set Q and the found paths stored in set F are eventually returned (line 14).

Table 6-2 Phase I Algorithm

```
   INPUT G(N,E), Q
   OUTPUT Q, F
 1:  F ← ∅
 2:  for each q=<o, g_o, l_o> ∈ Q do
 3:    if e=(s_o, t_o) ∉ E OR g_o=0 then continue
 4:    n_o ← ceil (g_o / B)
 5:    n_s ← getNumUnusedTransponders(s_o, P+)
 6:    n_t ← getNumUnusedTransponders(t_o, P-)
 7:    n ← min{n_o, n_s, n_t}
 8:    allocateTransponders(e, n)
 9:    f ← SP(G, o, g_o)
10:    if f ≠ ∅ then
11:      allocate(G, f)
12:      F ← F ∪ {f}
13:      g_o ← g_o − f.b
14:  return <Q, F>
```

The second phase is similar to the first phase, but for flows g_o through non-existing direct vlink. New capacitated direct vlinks are thus, added to the topology to support those flows.

In the third phase, the current topology is extended to a full mesh topology by adding uncapacitated vlinks (lines 2-4 in Table 6-3). Next, a randomised routing procedure runs for a given number of iterations (lines 6-31); at every iteration, the initial extended topology and the unserved bitrate are cloned, and the latter randomly sorted, giving higher priority to flows with higher remaining traffic (lines 7-12). Those flows with unserved bitrate are routed using one single path (lines 13-16). Aiming at minimizing the number of used transponders, link metrics are set proportional to the number of transponders needed to allocate the remaining capacity of the current flow (line 15). If no path is found, the corresponding cost is added to the iteration cost (lines 17-19); otherwise, in case the path capacity does not serve the remaining bitrate, we check whether the capacity of its links can be increased using the available resources, i.e., transponders in the end nodes and spectral resources to create a lightpath at the optical layer (lines 20-22). Finally, the path is allocated, and the remaining bitrate of the flow is updated as well as the iteration cost (lines 23-26). Once a solution has been built, a local search procedure is executed (line 27) aiming at finding a local minimum. The best topology and the found paths are returned as final solution (lines 29-32).

Table 6-3 Phase III Algorithm

```
INPUT G(N, E), U, thr
OUTPUT G*(N, E*), F
 1: G*(N, E*) ← G(N, E); F ← ∅
 2: for each e=(i, j) ∉ E | i, j ∈ N, i ≠ j do
 3:    E* ← E* ∪ {e}
 4:    setCapacity(e, 0)
 5: bestCost ← +∞
 6: for ite = 1 .. maxIter do
 7:    iteCost ← 0
 8:    G_ite ← G
 9:    U_ite ← U
10:    F_ite ← ∅
11:    for each u ∈ U_ite do u.order ← rand(0,1) * u_o
12:    sort(U_ite, u.order, DESC)
13:    for each u ∈ U_ite do
14:       if u.u_o = 0 then continue
15:       updateMetrics(G_ite, u.u_o)
16:       f ← SP(G_ite, u.o, u.u_o)
17:       if f = ∅ then
18:          iteCost ← iteCost + u.u_o·(|P|+1)
19:          continue
20:       if f.b < u.u_o AND canIncreaseCap(f, G_ite, u.u_o) then
21:          increaseCap(f, E_ite, u.u_o)
22:          f.b ← u.u_o
23:       F_ite ← F_ite ∪ {f}
24:       allocate(G_ite, f)
25:       u.u_o ← u.u_o − f.b
26:       iteCost ← iteCost + u.u_o·(|P|+1)
27:    <G_ite, F_ite> ← doLocalSearch(G_ite, F_ite)
28:    iteCost ← iteCost + numUsedTransponders(G_ite)
29:    if iteCost < bestCost then
30:       bestCost ← iteCost
31:       <G*, F> ← <G_ite, F_ite>
32: return <G*, F>
```

The local search procedure tries to reduce the total number of used transponders during the constructive phase. Since the number of transponders used in a node is computed as the maximum between transmission and reception, this procedure focuses on releasing transponders that actively contribute to that maximum at every node. The algorithm is detailed in Table 6-4, where all current vlinks in the VNT are processed (lines 2-20). The vlink with ports most actively contributing to the cost function is selected along with the set of paths routed through it (lines 6-7). This set is released from the VNT and sorted with respect to the bitrate (lines 8-11). Next, the set of paths is re-routed by possibly using new vlinks at zero objective cost (lines 13-15). In case of a feasible solution, the VNT is updated with these changes (line 20).

Table 6-4 Local Search Procedure

```
INPUT G(N,E), F
OUTPUT G*(N,E*), F*
 1:  G* ← G; F* ← F; E_rem ← E
 2:  while E_rem ≠ ∅ do
 3:    for each e ∈ E_rem do
 4:      e.balance ← computeTransponderBalance(e)
 5:    sort(E_rem, e.balance, DESC)
 6:    e ← removeFirst(E_rem)
 7:    F_e ← getPaths(e)
 8:    <G_aux, F_aux> ← <G*, F*>
 9:    release(G_aux, F_e)
10:    E_aux ← E_aux \ {e}
11:    sort(F_e, f.b, DESC)
12:    allRerouted ← true
13:    for each f ∈ F_e do
14:       recomputeZeroCostLinks(G_aux)
15:       f' ← SP(G_aux, o, f.b)
16:       if f' = ∅ then
17:         allRerouted ← false; break
18:       allocate(G_aux, f')
19:       F_aux ← (F_aux \ {f}) ∪ {f'}
20:    if allRerouted then <G*, F*> ← <G_aux, F_aux>
21:  return <G*, F*>
```

Table 6-5 presents the time complexity and running times of the proposed algorithms, where *R0* is the worst-case time complexity to find a feasible lightpath to support every direct vlink. For illustrative purposes, the computation time for the scenario with 14 nodes presented in section 0 is also provided. We studied the performance of the ILP formulation versus the proposed heuristic algorithm in a set of network instances with k_o=2 and 10 nodes, where solutions for the ILP could be obtained using a commercial solver. The heuristic algorithm produced solutions within one minute of computation with cost 1% higher than those produced by solving the ILP formulation in not less than 4 hours. This indicates the good trade-off between complexity and optimality of the proposed heuristic, which will be evaluated in more detail in Chapter 7.

Table 6-5 Time complexity and running times of the algorithms

Phase I/II	Phase III (per iter)	Local Search
$O(\|N\|^2 \cdot R0)$	$O(\|N\|^2 \cdot (\|E\| \cdot \log\|N\| + R0))$	$O(\|E\|^2 \cdot (\log\|E\| + \|Fe\| \cdot \log\|N\|))$
<1ms	298ms	234ms

The previous algorithms provide some basic procedures to solve the VENTURE problem. As we will see in-depth in the next chapter, these procedures can be used in combination with advanced metaheuristic algorithms for combinatorial network optimisation. Among all the solving methods considered including those above, let

us advance the reader that Iterated Local Search (ILS) appears to provide the best performance among all, and hence we fix it as the selected VENTURE algorithm for the following numerical study.

6.4 Numerical results

6.4.1 Simulation scenario

For evaluation purposes, we ran simulations in iONESim. To measure the effect of volumetric and directional changes in traffic, we implemented generators that inject traffic following the Users and Datacenter pre-defined traffic profiles following the traffic generation set up presented in Section 5.4.1.

Finally, the set of nodes was divided into two subsets to generate changes in the direction of the traffic; ODs with destination one of the nodes in the first subset follow the *Users* profile, while the others follow the *DC2DC* one. We consider a scenario where a maximum of 26 × 100 Gb/s transponders per node is equipped. With such configuration, the static and threshold-based approaches are applied to a full-mesh 14-node VNT, where the initial capacity of each vlink ranges from 100 to 200 Gb/s. To obtain OD traffic prediction, the ANN models presented in Chapter 5 were trained applying the fitting algorithm in Fig. 5-4 on a training dataset with modelled data belonging to the last weeks. Results presented are the average of 100 runs with different random number generation seeds.

Next, we compare the effect of the unserved traffic and the number of used transponders under the following approaches:

- A threshold-based VNT reconfiguration approach, running continuously with a vlink utilisation threshold of 90%.
- The VENTURE algorithm triggered at fixed intervals of one hour.
- For the sake of completeness, we also include the static VNT approach where no reconfiguration is performed. This VNT is computed by running the VENTURE algorithm using as input the maximum predicted traffic matrix for the whole simulated period, thus obtaining a full-mesh topology.

6.4.2 Blocking probability performance

Let us now analyse the obtained blocking probability for the range of loads considered, where a value of 1 represents a reference traffic matrix. Values for both, the static and the threshold-based approaches are omitted since yield zero blocking probability.

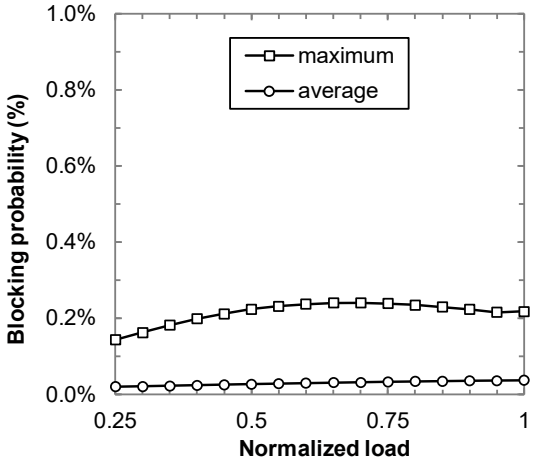

Fig. 6-6 Average and maximum hourly blocking probability of VENTURE vs load.

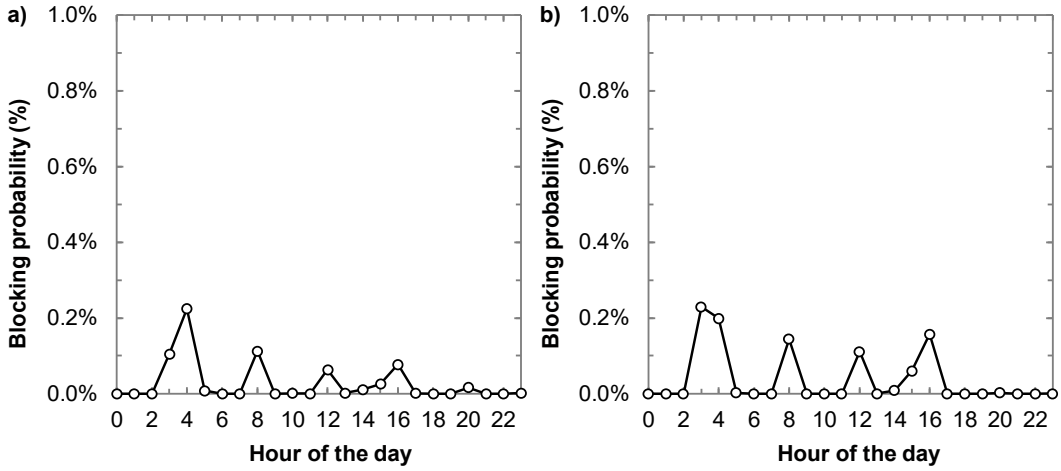

Fig. 6-7 Blocking probability along one day of VENTURE for a normalised load of 0.48 (a) and 1.0 (b).

In the case of the VENTURE approach, Fig. 6-6 plots the average and maximum hourly blocking along a day. We observe that, for a wide range of traffic loads, maximum blocking probability is below 0.24%, while that on average is virtually zero. Fig. 6-7a-b analyse the evolution of blocking probability during the day for a low and a high load, respectively. We observe that small peaks of blocking probability appear related to abrupt changes in the injected traffic and last for a couple of hours at the most, which is the time that VENTURE takes in fully adapting the VNT to traffic changes with the specific configuration selected.

6.4.3 Transponder utilisation performance

Let us now focus on the use of transponders. Fig. 6-8 plots, for each approach, the maximum transponder usage as a function of the load. Both, the static and the threshold-based approaches show a constant transponder usage for loads lower than 0.5, which is increased from that load up. For low loads, the capacity of vlinks in the fully meshed VNT is 100 Gb/s in both cases, and it is increased to 200 Gb/s for high loads under the static approach. The threshold-based approach, however, is able to manage the use of transponders by flexibly using available transponders to increment the capacity of vlinks running out of capacity; this way it achieves transponder savings up to 11% with respect to the static VNT approach.

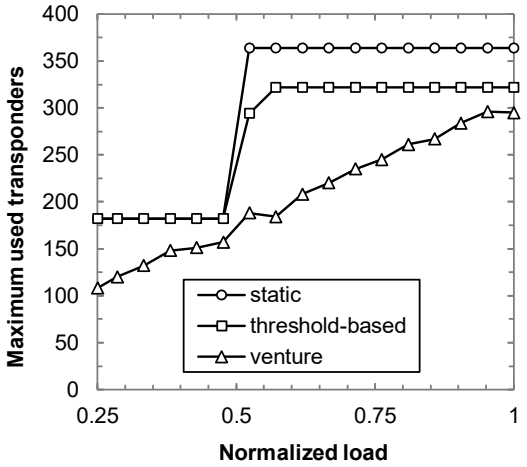

Fig. 6-8 Maximum used transponders vs load.

Interestingly, transponder usage scales linearly with the load with VENTURE. Compared to the threshold-based approach, VENTURE obtains savings between 8% and 42%. Fig. 6-9a-b focus on the use of transponders along the day for a low and high load for the three approaches. Apart from the constant transponder usage in the static approach, we show the different usages of the threshold-based and the VENTURE approaches. In particular, we observe how the VENTURE approach is able to remarkably reduce up to 45% transponder usage at some hours, mainly when the DC2DC traffic profile is dominant. On the other hand, in those hours when Users traffic profiles dominate, transponder usage under VENTURE still outperforms that of the threshold-based approach.

In conclusion, the VENTURE approach maximises the overall utilisation of available transponders in two different ways: *i*) by reconfiguring the virtual topology to follow traffic direction changes, and *ii*) by increasing the capacity of vlinks when the traffic increases.

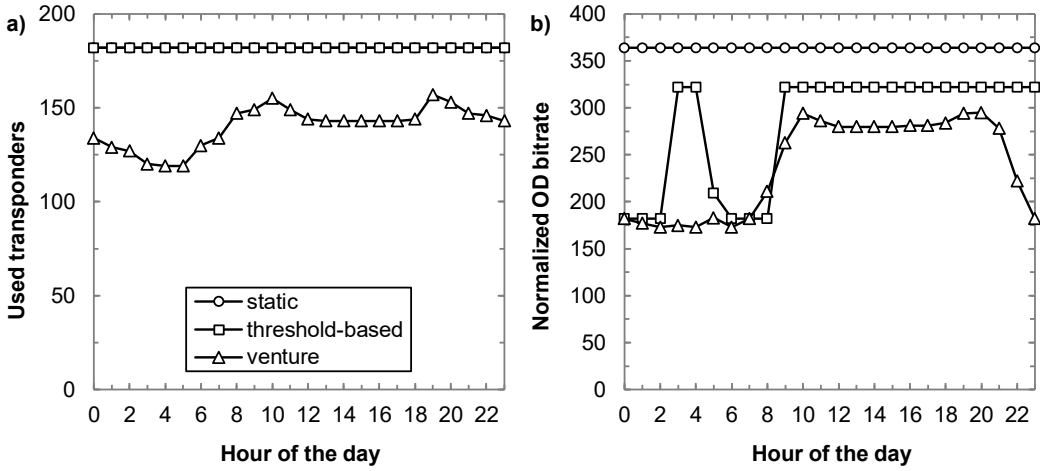

Fig. 6-9 Daily transponder usage for a normalised load of 0.48 (a) and 1.0 (b).

6.5 Conclusions

An efficient approach, named as VENTURE, to adapt the current VNT to future traffic conditions aiming at minimising OPEX has been proposed. The approach consists in monitoring OD traffic in the IP/MPLS routers and applying data analytics to learn predictive models that are used as inputs of a reconfiguration problem. In particular, an ANN for every OD pair was proposed as a predictive model along with an algorithm to obtain a highly accurate ANN using as few coefficients as possible. The VENTURE reconfiguration problem was formally stated and formulated as an ILP. Given its complexity for valid short-term solutions, a heuristic algorithm to provide near-optimal solutions in practical computation times was proposed.

We compared the performance of VENTURE through simulation against the static and the threshold-based approaches. We observed savings between 8% and 42% in the number of transponders to be installed in the routers when the VENTURE approach was applied. Also, VENTURE can deactivate transponders during low traffic hours thus, decreasing the energy consumption and releasing lightpaths from the underlying optical layer, which contribute to a costs reduction.

Chapter 7

Advanced VENTURE algorithms

The VENTURE problem presented and solved in Chapter 6 is an application of cognitive in-operation network planning, where data analytics (prediction) is combined with mathematical optimisation to adapt the VNT to the near future. The problem was mathematically formulated, and a heuristic algorithm devised to produce quality solutions in practical times. The proposed heuristic algorithm was evaluated against purely reactive approaches offering considerable savings regarding needed transponders.

Despite the outperforming results of the VENTURE algorithm against purely reactive VNT reconfiguration, the question *how well does the VENTURE algorithm perform* remains to be answered formally. In other words, an assessment of the VENTURE algorithm's performance including, among others, optimality study against exact solutions and a comparison against competitive, state-of-the-art heuristics is needed. In particular, it would be interesting to see if the inclusion of traffic prediction in other solving methods could provide better solutions than our proposal.

To that end, we first show the application of two metaheuristics to solve the VENTURE problem; we propose two state-of-the-art, neighbourhood-based metaheuristics selected for their proven performance in solving combinatorial network optimisation problems. Both metaheuristics are adapted to the VENTURE problem use case. Secondly, a numerical study is performed to assess the quality of the solutions provided by the original approach, by comparing them against those obtained using two proposed metaheuristics and also against those obtained by using an exact solving method.

7.1 GRASP heuristic algorithm

In order to adapt the GRASP metaheuristic to solve the VENTURE problem, we need to provide the definitions of candidate and the GCF. A candidate for the VENTURE problem is defined as an OD pair with some remaining capacity to be allocated in the VNT. In particular, this is formally defined in lines 10-11 of VENTURE algorithm's main pseudocode (Table 6-1). For the sake of clarity let us refresh the definition provided there: a candidate u is a tuple $<o, u_o>$, where o represents an OD pair and u_o the capacity that remains to be allocated in the VNT.

Regarding the GCF, the pseudocode in Table 7-1 details how the greedy cost of a candidate u can be computed during the execution of the VENTURE-based GRASP heuristic. Let us first denote the current solution x as the tuple $<G(N,E), F>$ containing the current VNT with a set of allocated paths F thus following the same notation of Chapter 6. The algorithm starts by updating the vlink metrics of the extended topology as a function of the capacity to be allocated (lines 1-9). A low metric is set if a vlink has enough capacity to allocate the needed bitrate (lines 3-4), whereas a high metric is used otherwise. In the case that a vlink has not enough capacity to allocate u_o we distinguish between two cases: a high metric is set in case that the vlink capacity can be increased (with the corresponding transponder utilisation) or the link metric is otherwise forbidden (lines 5-9). Once the vlink metrics are updated the shortest path is computed for OD pair o (line 10), and its total metric is eventually returned as the greedy cost (lines 11-14).

Table 7-1: GCF for GRASP-based VENTURE

```
   INPUT u=<o, u_o>, x=<G(N,E), F)
   OUTPUT cost
 1:  for e ∈ E do
 2:     b ← unusedCapacity(e)
 3:     if b ≥ u_o then
 4:        e.metric ← 1
 5:     else
 6:        if canIncreaseCap(e, u_o - b) then
 7:           e.metric ← |E|
 8:        else
 9:           e.metric ← +∞
10:  f ← SP(G, o)
11:  cost ← 0
12:  for e ∈ f do
13:     cost ← cost + e.metric
14:  return cost
```

Since the computational complexity of updating the vlink metrics is $O(|E|)$ and it is lower than that of the selected shortest path algorithm (Dijkstra's), the worst-case time complexity of the previous GCF is that of the shortest path algorithm: $O(|E|+|N| \cdot \log(N))$. That said, the worst-case time complexity needed to

recompute the whole candidate list (line 4 in Table 2-2) is $O(|OD| \cdot (|E|+|N| \cdot \log(N)))$.

Finally, the local search procedure used in this implementation of GRASP is the same presented in Table 6-4 of Chapter 6 for the original VENTURE algorithm. Notice that this implementation of GRASP starts from a partial solution provided during phases I and II of the algorithm. Therefore, although GRASP produces multiple solutions all of them will share the part deterministically built in the two first phases, possibly leading to a limited exploration of the feasible region (independently of the α parameter). To overcome this possibility, we next present an alternative approach which allows overcoming this problem.

7.2 ILS heuristic algorithm

Let us now present the adaptation of ILS used in replacement for the third phase VENTURE algorithm. As explained before, to use ILS we need to provide a constructive, a local search and perturbation procedures. The local search procedure considered for ILS is the one already presented in Table 6-4, also used for GRASP.

As for the constructive phase, Table 7-2 presents the considered procedure. A major part of the procedure is based on the original third phase algorithm presented in Chapter 6 (Table 6-3). However, in this case, only a single greedy solution is built instead of performing a greedy-randomised multi-start search. The procedure starts by extending the current topology to a full mesh by creating zero-capacity vlinks (lines 1-4). Next, the set of unserved OD capacities is sorted greedily, by placing first those OD pairs with higher unserved bitrate (line 5). Once the set of OD pairs is sorted, an iterative procedure is run to process the sorted list of OD pairs aiming at allocating their remaining capacity in the VNT (lines 6-17). At each iteration, the vlink metrics are adapted in order to minimise the transponders needed to allocate the remaining bitrate of a given OD pair (line 8). This minimum cost path is obtained by computing the shortest path algorithm (line 9) similarly to the GCF defined for GRASP (Table 7-1); observe that an empty shortest path can be returned if there is not enough capacity left in the VNT. In case that a path with insufficient bitrate is found we check whether the traversed vlinks can increase their capacity by using additional transponders (lines 12-14). The path is allocated and added to the set of paths F (lines 15-16). After processing all the OD pairs, the final solution is returned (line 18).

The computational complexity of this constructive procedure is $O(|N|^2 \cdot (|E| \cdot \log|N| + R0))$, where $R0$ is the worst-case time complexity to find a feasible lightpath to support every direct vlink. Note that this is equivalent to a single iteration of the original third phase algorithm presented in Chapter 6.

Table 7-2: Constructive procedure for ILS-based VENTURE

```
INPUT G(N, E), U, thr
OUTPUT x = <G*(N, E*), F>
 1:  G*(N, E*) ← G(N, E); F ← ∅
 2:  for each e = (i, j) ∉ E | i, j ∈ N, i ≠ j do
 3:     E* ← E* ∪ {e}
 4:     setCapacity(e, 0)
 5:  sort(U, u.uo, DESC)
 6:  for each u ∈ U do
 7:     if u.uo = 0 then continue
 8:     updateMetrics(G*, u.uo)
 9:     f ← SP(G*, u.o, u.uo)
10:     if f = ∅ then
11:       continue
12:     if f.b < u.uo AND canIncreaseCap(f, G*, u.uo) then
13:       increaseCap(f, E*, u.uo)
14:       f.b ← u.uo
15:     F ← F ∪ {f}
16:     allocate(G*, f)
17:  return <G*, F>
```

The perturbation procedure that is applied to a solution at each iteration of ILS is presented in Table 7-3. The procedure consists of two blocks: solution destruction and solution restoration blocks. During the destruction block, the current solution is broken by removing a random subset of capacitated vlinks from the VNT (lines 1-9). The amount of removed vlinks is determined by the value of the strength parameter (i.e., a percentage of current vlinks). For each removed vlink, the set of paths traversing the vlink is retrieved (line 5) and all of them released from the VNT (line 7); a set U is used to keep track of the released paths (line 8). Finally, the capacity of each removed vlink is set back to zero (line 9), thus allowing a possible re-utilisation in the future. During the solution restoration block, all the released paths from the VNT are re-allocated following a greedy constructive approach, the same than the one presented for the constructive procedure in Table 6-3. In this case, however, the number of processed elements in U is not fixed and depends on the perturbation strength.

An important aspect of ILS-based VENTURE is its capability to perform a broader exploration of the solution space in contrast with both the original VENTURE algorithm and GRASP-based VENTURE. The explanation to this can be found in the perturbation procedure: the procedure does not only re-allocate those elements $u \in U$ used as input for the third phase but in general any $u = <o, u_o>$ affected by the removal of a vlink, thus opening the possibility of re-allocating OD pair capacity pre-fixed during phases I and II of the algorithm.

Table 7-3: Perturbation procedure for ILS-based VENTURE

```
INPUT x = <G(N, E), F>, strength
OUTPUT x^pert = <G*(N, E*), F>
 1:  G*(N, E*) ← G(N, E); F ← ∅
 2:  E^rem ← selectAtRandom(E, e.b > 0, strength)
 3:  U ← ∅
 4:  for e ∈ E^rem do
 5:     F_e ← getPathsTraversing(e)
 6:     for f ∈ F_e do
 7:        release(f)
 8:        U ← U ∪ u = <f.o, f.b>
 9:     setCapacity(e, 0)
10:  sort(U, u.u_o, DESC)
11:  for each u ∈ U do
12:     if u.u_o = 0 then continue
13:     updateMetrics(G*, u.u_o)
14:     f ← SP(G*, u.o, u.u_o)
15:     if f = ∅ then
16:        continue
17:     if f.b < u.u_o AND canIncreaseCap(f, G*, u.u_o) then
18:        increaseCap(f, E*, u.u_o)
19:        f.b ← u.u_o
20:     F ← F ∪ {f}
21:     allocate(G*, f)
22:  return <G*, F>
```

7.3 Numerical results

In this section we present the reference instances used to compare the VENTURE algorithm against the presented GRASP and ILS -based VENTURE versions, also including an exact solving method for the sake of a clearer evaluation.

7.3.1 Instance generation

Aiming at extensively evaluating the various algorithms for solving the VENTURE problem we devised an instance generator capable of generating off-line instances by tunning a set of parameters. Recall from Chapter 6 that an instance of the VENTURE problem consists of the current and the predicted traffic matrices (i.e., D and OD) along with the current VNT; for the sake of simplicity, we considered an empty topology as initial VNT. Once the number of network nodes $|N|$ is defined (and so the sizes of D and OD) two different sets of parameters allow specifying how both traffic matrices are generated. For the generation of D we must define (*i*) the total bitrate in the traffic matrix (in Gb/s), (*ii*) a subset of OD pairs (called *hubs*) concentrating a given amount of the total bitrate and (*iii*) a *roughness*

parameter that specifies the variability between the bitrate of different OD pairs. Once these three parameters are set, the generation of matrix D is as follows. Initially, a fraction of the total bitrate is assigned to the subset of *hub* OD pairs. The remaining bitrate is then distributed among $|N| \cdot (|N|-1)$ bins using a truncated normal distribution centred at zero with variance inversely proportional to *roughness*: a lower roughness value causes the bitrate to be uniformly distributed among the bins whereas a large value causes the opposite. Once the remaining bitrate is distributed across the bins, the bitrate of each bin is mapped to an entry in matrix D at random. Fig. 7-1 illustrates how choosing different roughness values can lead to the generation of different traffic matrices. The matrices are represented by a heat map, where dark (clear) represents high (low) OD bitrate.

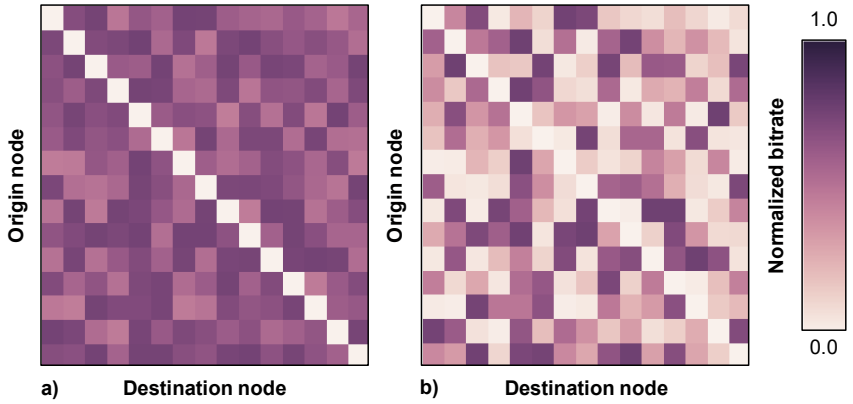

Fig. 7-1 Generation of matrix D for a 15-node VNT: low (a) and high (b) roughness values used for the bitrate distribution.

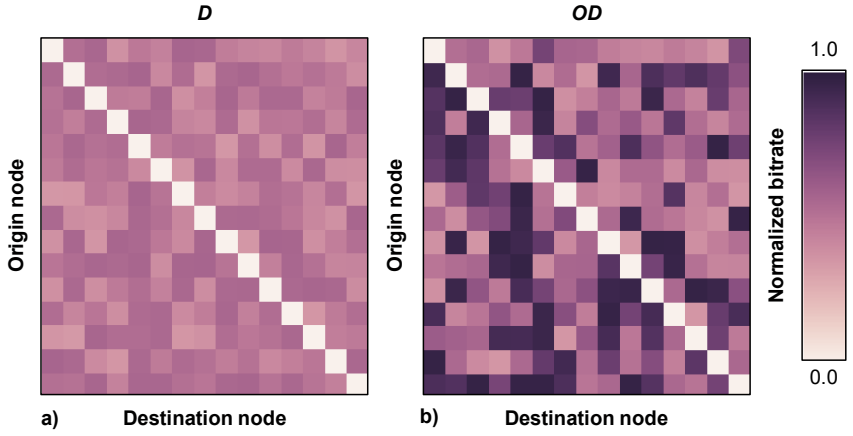

Fig. 7-2 Approximately 50% of the OD pairs in D increase their capacity a 70%.

Once matrix D is computed, it is used as input for the generation of matrix OD by modifying OD bitrate in terms of traffic direction and volume. Directional traffic

variations can be introduced in matrix *D* by allowing a particular subset of OD pairs to increase (or decrease) their bitrate. To that end, a subset of candidate OD pairs is defined to increase or decrease their bitrate along with a percentage specifying the relative increment or decrement. A uniform probability can be adjusted to decide which candidate OD pairs are selected for incrementing or decrementing their entry in matrix *OD*. For illustrative purposes, Fig. 7-2 shows an example pair <*D*, *OD*>. In one hand, OD bitrate in matrix *D* is uniformly distributed (i.e., no hubs and a low roughness). On the other hand, matrix *OD* is generated by increasing each OD pair's bitrate a 70% percent with probability 50% of being selected. Note that a specific subset of candidate OD pairs can also be specified to have more control over the generation of matrix *OD*.

Using the generator above, a set of 12 reference instances representing different traffic scenarios of a 15-node VNT were generated. We assume that an unlimited number of B = 100 Gb/s transponders are available in the optical layer. We chose this network size to facilitate the obtention of exact solutions, therefore allowing a more detailed performance analysis of the different VENTURE heuristics. These instances are divided into three groups depending on the bitrate *granularity* of each OD pair with respect to *B*. A *high* granularity is defined to represent traffic scenarios where bitrate values are below *B,* a*verage* granularity to represent OD bitrate centered around *B*, and finally *low* granularity for traffic scenarios with OD bitrate above *B*.

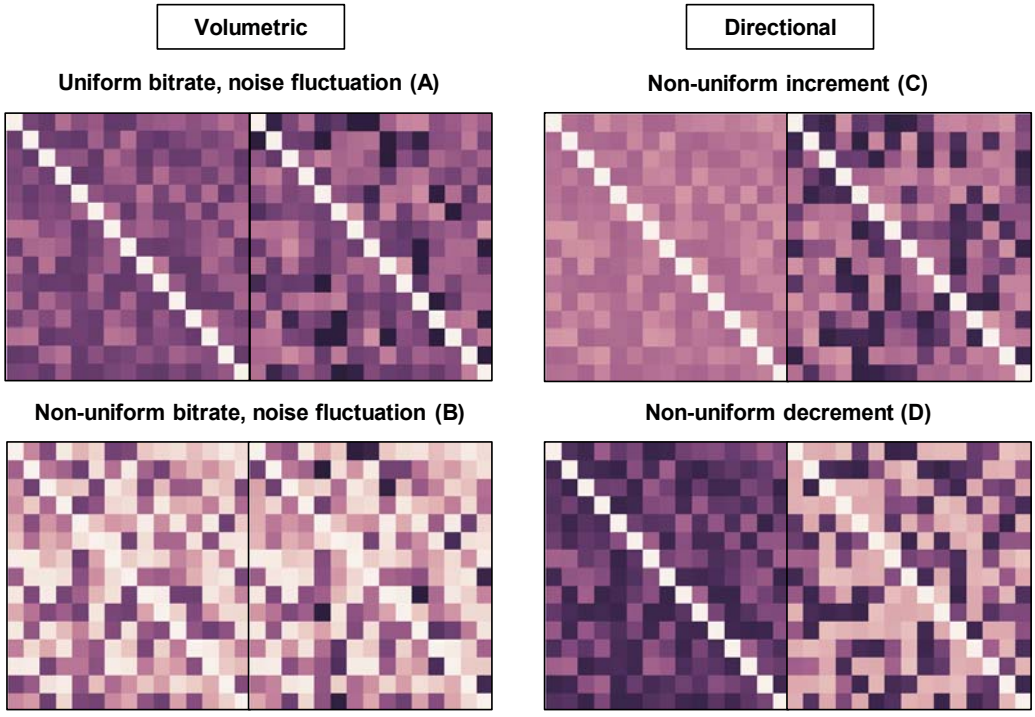

Fig. 7-3 Different <*D*, *OD*> traffic transitions for a given granularity value.

These three scenarios are selected since they induce a different combinatorial complexity for the computation of paths serving *D* and *OD*. For each granularity value, the four instances shown in Fig. 7-3 are defined to represent four different traffic scenarios. Among them, two of the instances represent uniform and non-uniform volumetric variation (A and B) whereas the other two represent directional traffic variation (C and D).

7.3.2 Performance evaluation

The performance of the different solving methods for the VENTURE problem was evaluated by solving the previous set of instances. Specifically, we compare the performance of 4 solving methods, 3 of which are heuristic algorithms (the VENTURE algorithm from Chapter 6 hereafter referred to as *base* VENTURE, GRASP-based VENTURE and ILS-based VENTURE) and the fourth is an exact method (*branch and bound*). The three heuristics were implemented in C++ whereas Branch and Bound (B&B) is based on CPLEX v.12.5.1 commercial solver. For the particular case of parameter-based metaheuristics (GRASP and ILS), different values of their parameters are studied. In addition, 100 repetitions of each instance and algorithm configuration pair were run with different random seeds. The stopping criterion for the heuristics is a maximum running time of 1 minute, whereas for CPLEX the maximum running time is set to 4 hours to facilitate the optimality gap estimation. All solving methods were executed on a computer with an Intel Core i7-4790K processor, 16GB of RAM and Ubuntu 16.04.

Table 7-4 shows the objective cost value of the four different solving methods. For GRASP and ILS, we show the results obtained with the best parameter configuration. For B&B, the best linear relaxation objective cost obtained during the execution of B&B is also shown; we use it to estimate the absolute and relative optimality gaps of each solving method. We can observe that all the heuristic algorithms present an optimality gap between 8% and 9%. By looking at the table, we observe a larger gap in all the methods for those instances with high bitrate granularity. This can be explained as a result of the higher granularity of the OD bitrate with respect to *B*: more MPLS paths can be allocated in the same vlink in contrast with low granularity instances requiring more capacity increments to perform such grooming. This induces more routing combinations to serve matrices *D* and *OD* at a low cost, hence more complexity in finding quality solutions. By analysing the average and low granularity instances, we detect that the gap effectively decreases. Although for high granularity instances B&B outperforms the heuristics in terms of solution quality, these latters are capable of finding quality solutions in only one minute. For average and low granularity instances at least one heuristic approach outperforms B&B both regarding optimality and running time, being ILS the best performing heuristic.

Table 7-4 Objective cost and optimality gap of VENTURE solving methods.

Instance		Objective function cost					Optimality gap (%)			
Granul.	Type	B&B (4 hours)		Heuristics (60 seconds)			B&B	Base	GRASP	ILS
		Best bound	Best integer	Base	GRASP	ILS				
High	A	142.00	165	167.37	164.91	165.63	16.20%	17.87%	**16.13%**	16.64%
	B	190.75	212	212.08	214.40	213.25	**11.14%**	11.18%	12.40%	11.80%
	C	166.69	187	187.56	186.50	187.42	12.18%	12.52%	**11.88%**	12.44%
	D	136.10	148	157.98	153.81	154.28	**8.74%**	16.08%	13.01%	13.36%
Average	A	230.74	250	250.81	255.46	249.23	8.35%	8.70%	10.71%	**8.01%**
	B	357.80	377	376.92	379.77	377.70	5.37%	**5.34%**	6.14%	5.56%
	C	291.07	315	314.57	320.24	313.55	8.22%	8.07%	10.02%	**7.72%**
	D	217.08	235	235.53	237.83	233.53	8.26%	8.50%	9.56%	**7.58%**
Low	A	455.44	484	484.14	488.25	480.99	6.27%	6.30%	7.20%	**5.61%**
	B	721.24	745	746.06	749.59	745.81	**3.29%**	3.44%	3.93%	3.41%
	C	590.00	618	618.98	624.15	616.65	4.75%	4.91%	5.79%	**4.52%**
	D	426.22	457	457.26	460.89	453.56	7.22%	7.28%	8.13%	**6.41%**

Next, we study in detail the parameter tuning for GRASP. In particular, Fig. 7-4 shows the absolute optimality gap variation that is obtained when GRASP-based VENTURE is executed and the solutions obtained compared against those produced by base VENTURE, as a function of the α parameter. Exactly as before, an estimation of the absolute optimality gap is computed using the best linear relaxation objective cost obtained during B&B execution. Although for average and low granularity instances it is immediate to see that GRASP does not improve the performance of the solutions obtained by base VENTURE, results show different behaviour for high granularity instances. If we focus on these, we can see that in three out of four traffic transitions (A, C and D) GRASP-based VENTURE is able to decrease the optimality gap between 5% and 20% when compared to base VENTURE. We observe a typical behaviour in the tunning of the α parameter, where a value of $\alpha \in [0.4, 0.6]$ produces the best tradeoff between greediness and randomness.

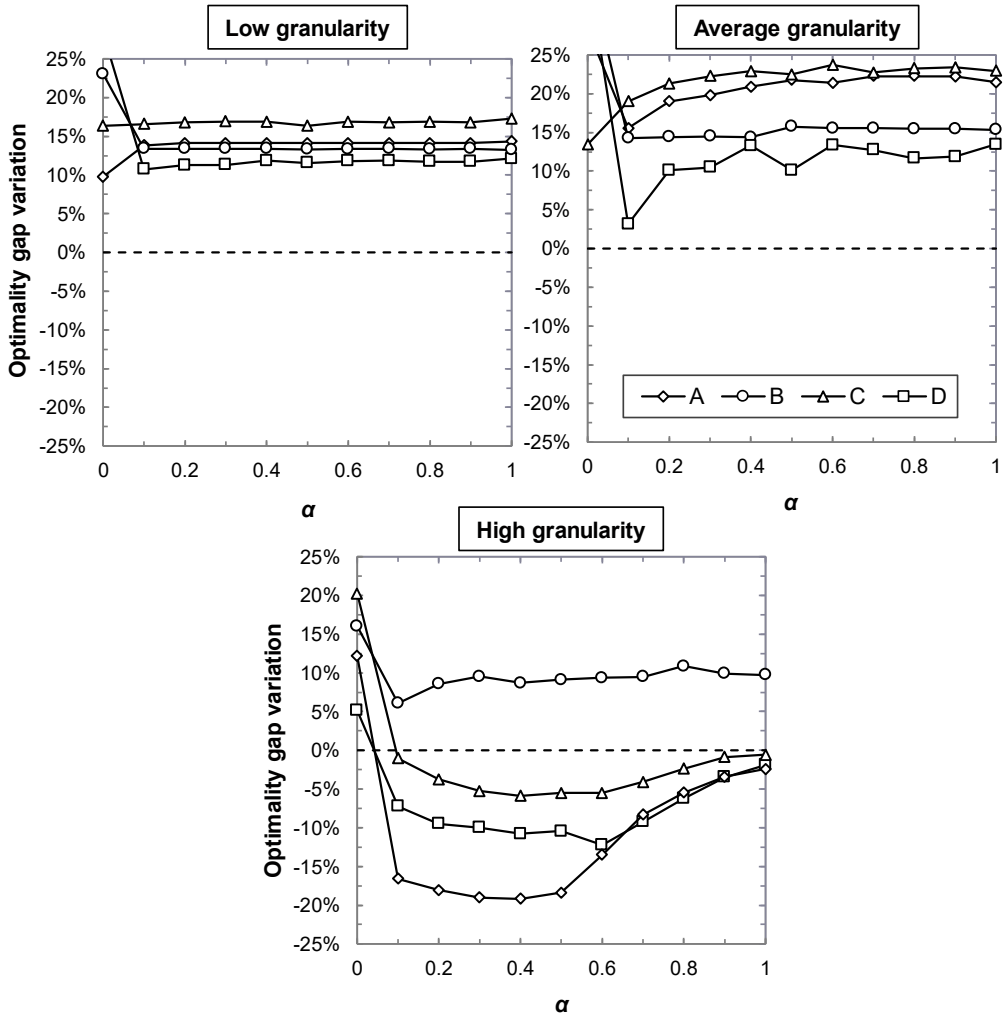

Fig. 7-4 Gap reduction of GRASP-based VENTURE w.r.t. base VENTURE for different values of the α parameter.

Similarly, Fig. 7-5 shows the optimality gap variation when ILS-based is run against base VENTURE. In contrast with GRASP, ILS is able to decrease the gap obtained with base VENTURE for any traffic granularity, with the exception of B-type instances (non-uniformly distributed OD bitrate) where base VENTURE finds better solutions. In those instances where ILS finds better solutions we obtain an optimality gap reduction between 5% and 20%. Note that as a result of the quality of the base VENTURE algorithm, a 20% decrement in terms of optimality gap translates in only 2% decrement of objective cost value (used transponders). Regarding the best parameterisation of ILS, it is worth observing that the optimal strength parameter value seems to present dependency on the OD bitrate granularity. For low granularity instances, it is better to apply aggressive perturbations (60%-70% of vlinks removed), for average granularity instances the

optimal strength is decreased (50%-60% of vlinks removed) reaching the most conservative perturbation strength for high granularity instances (10%-30% of vlinks removed). In light of these results, let us focus only on ILS-based VENTURE for the last part of this study, as it appears as the best candidate.

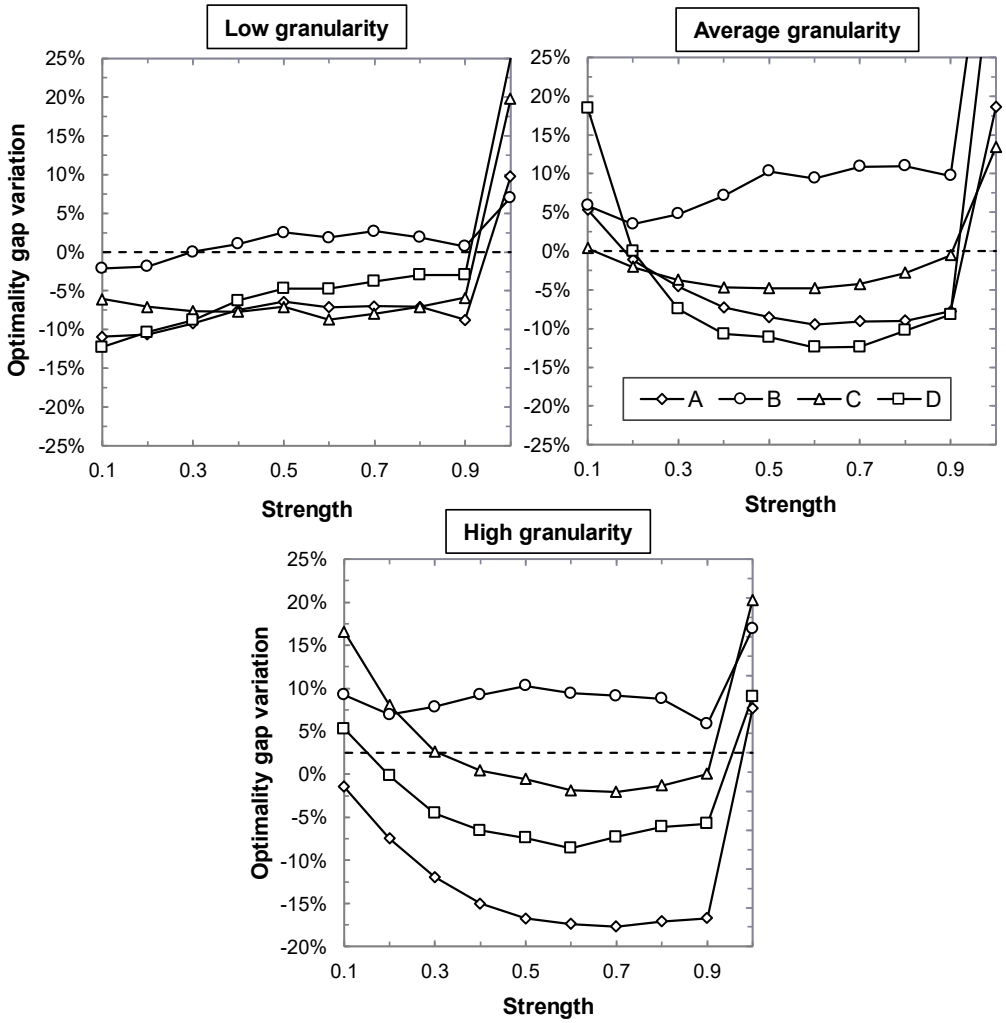

Fig. 7-5 Gap reduction of ILS-based VENTURE w.r.t. base VENTURE for different values of the perturbation strength parameter.

Although the evaluation of the heuristics above for VENTURE is done by setting a maximum running time of 1 minute, we are interested in knowing how the search of solutions evolves during an execution. Or in other words, to study how much time is needed by each of the heuristics to reach a comparable solution quality. To that end, we make use of a data visualisation technique known as *time-to-target plot* (tttplot) [Ai07]. Given a target objective cost value *C*, a tttplot shows an estimation of the cumulative probability distribution of the random variable *time to target objective cost C*. By analysing these plots, we can draw conclusions about the

convergence of base and ILS -based VENTURE to a common target cost under different instances. For the next tttplots, the least objective cost obtained among the compared heuristics is used as a target.

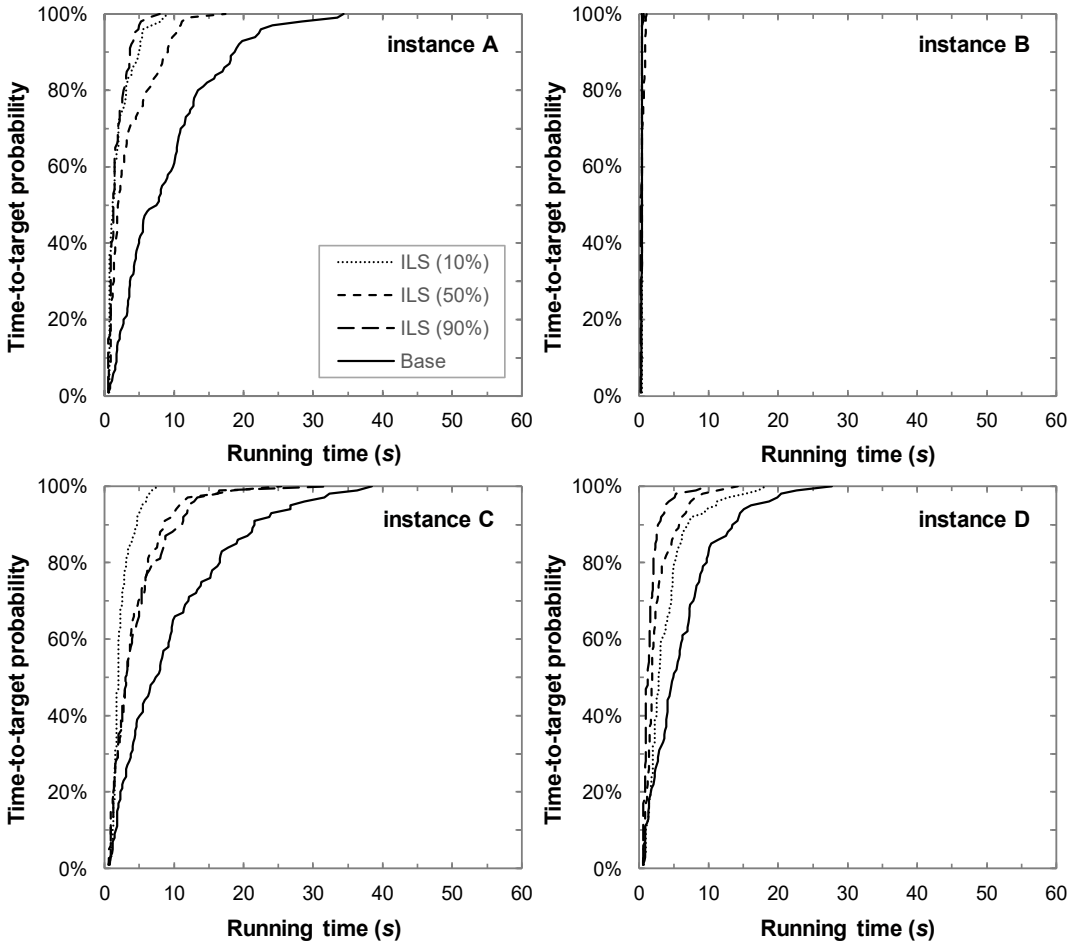

Fig. 7-6 Running time vs time-to-target probability of ILS-based and base VENTURE for low granularity instances.

Fig. 7-6 shows the tttplots corresponding to the execution of base VENTURE and ILS-based VENTURE with perturbation strengths 10%, 50, and 90% in the subset of low granularity instances. We can observe that all the configurations of ILS-based VENTURE reach the target objective cost with 80% probability in no more than 10 seconds (6% of the total running time). This is in contrast with base VENTURE, which presents a slower convergence to the target objective cost: it reaches it with probability 80% in approx. 25% of the total running time. This does not hold for B-type instances, where all the heuristics seem to converge at the same rate to a given target cost. Note that in the previous figure there are not crossings between curves, meaning that we all the heuristics can be ordered from slower to faster convergence rate to an objective target cost. However, this rate does not

imply reaching a better final solution. In fact, if we collate these results with those in Fig. 7-5 for low granularity instances, we realise that the optimal configuration of ILS for the different instances does not reach a given target cost faster. Similar results are shown in Fig. 7-7 for instances with average OD bitrate granularity. In contrast with the previous results, in these instances, base VENTURE presents a shorter time-to-target than ILS for some parameter configurations and instances.

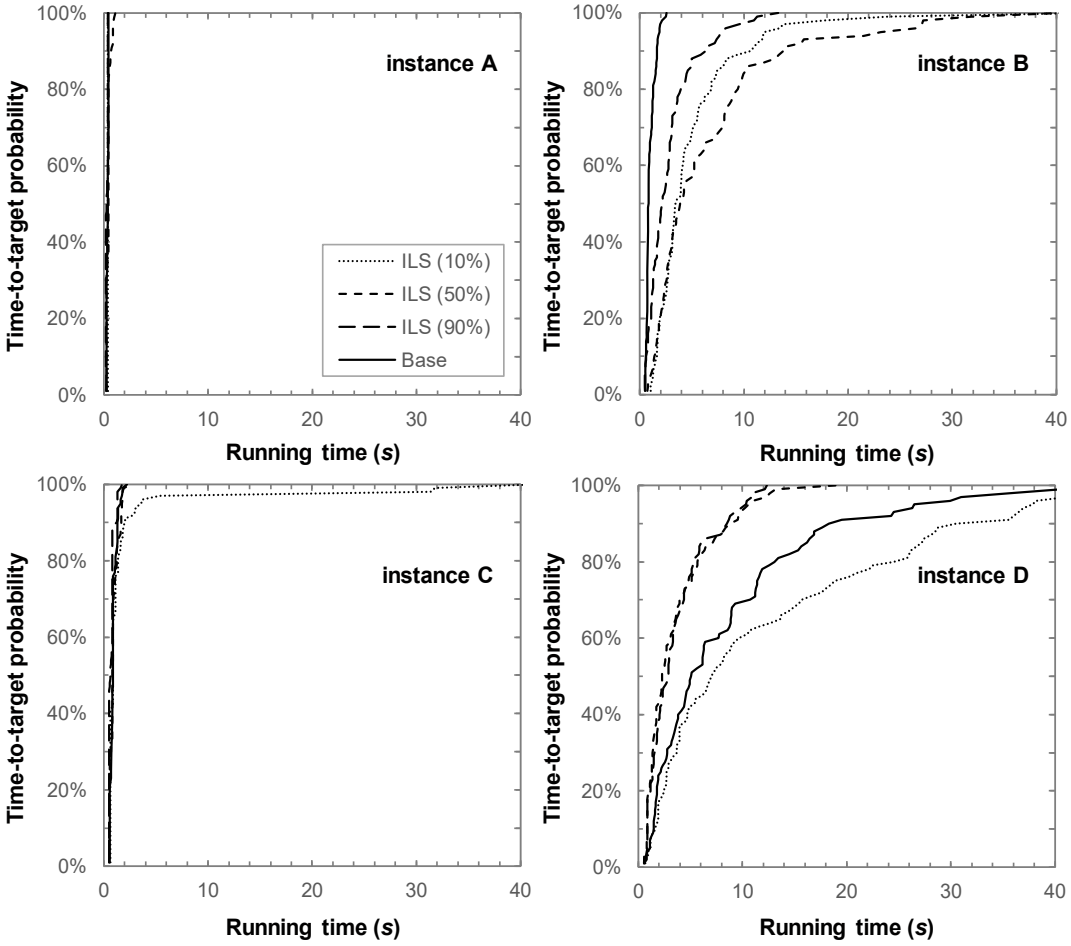

Fig. 7-7 Running time vs time-to-target probability of ILS-based and base VENTURE for average granularity instances.

7.4 Conclusions

The VENTURE problem was presented and solved in Chapter 6 by introducing a heuristic approach known as the VENTURE algorithm. This algorithm showed interesting savings in terms of used transponders in contrast with purely reactive VNT reconfiguration. Notwithstanding these results, it remained unclear whether

this algorithm was the best option among the possible heuristic approaches for solving the VENTURE problem.

To answer this question, GRASP and ILS metaheuristics have been presented and used to solve VENTURE. These two metaheuristics were chosen because of their proven performance to solve combinatorial network optimisation problems. As a result, we compared the performance of three different algorithms: the original VENTURE algorithm from Chapter 6, GRASP-based and ILS-based VENTURE. These three algorithms were also compared to an exact method using a commercial solver.

To perform an exhaustive evaluation of the different solving methods for VENTURE we developed an instance generator capable of producing instances with volumetric and directional traffic changes. A set of 12 instances were generated featuring different traffic changes and OD bitrate granularity with respect vlink capacity.

We observed that the heuristic approaches present a similar optimality gap of 9% on average. However, this gap decreases with the OD bitrate granularity, reaching a 5% optimality gap when solving those traffic scenarios. We also observed that the heuristics outperform branch and bound, being ILS the best candidate heuristic. Although ILS is capable of decreasing up to a 20% the optimality gap obtained with the original VENTURE algorithm, this translates in only 2% objective cost decrement. By analysing tttplots for ILS-based VENTURE and the original VENTURE algorithm, we observed a similar evolution towards quality solutions in their executions. Therefore, the correct performance of the VENTURE algorithm devised in Chapter 6 is validated.

Chapter 8

Core VNT reconfiguration based on metro-flow traffic prediction

In this chapter, we propose to aggregate metro-flow traffic models into OD traffic models for the core. Section 8.1 presents the drawbacks of using predictive OD traffic models based on monitoring the core, motivating the need of extending data analytics to the metro areas to obtain metro-flow predictive models that can be afterwards aggregated to obtain updated OD models in practical times. Based on aggregated models, we can predict future traffic for the OD pairs and use it as input for core VNT re-optimisation purposes.

The discussion is supported by the results presented in Section 8.3. The accuracy of OD predictive models based on metro-flow model aggregation is validated by means of exhaustive numerical results, as well as their validity for supporting core VNT reconfiguration based on traffic prediction under metro-flow rerouting is the network simulation scenario.

8.1 Flow traffic prediction under changing traffic

Modelling core OD and metro-flow traffic independently at each network segment can lead to degraded performance in the quality of the predictive models after metro-flow rerouting. OD pairs might aggregate many metro-flows and hence, reroute some of the metro-flows in the metro areas might change the aggregated traffic pattern of some ODs in the core network. For illustrative purposes, Fig. 8-1 presents an example of a metro-flow (*mf1*) originated at some metro network and routed toward datacenter DC2 through the core VNT. The metro-flow is

encapsulated in a MPLS Label Switched Path (LSP) and routed through the VNT. Initially, the LSP enters the core VNT through ingress node R2 toward egress node R3, so it is aggregated into core OD pair R2→R3. Due to metro re-optimisation, the metro controller reroutes the metro-flow LSP so that to enter the core VNT through edge node R1, being aggregated into core OD pair R1→R3. As a result, the traffic profiles of both OD pairs (R1→R3 and R2→R3) have now changed.

Fig. 8-2 illustrates the traffic profile change in OD R2→R3. Before the rerouting event, the predictive core OD model perfectly fits the actual traffic; once metro-flow LSP *mf1* has been rerouted, the OD traffic profile changes and the corresponding predictive model becomes obsolete, thus triggering a re-estimation based on new monitoring data. Note that the difference between the actual traffic volume and the obsolete prediction can be mistakenly confused with an OD traffic anomaly [AV17-1], which would trigger unnecessary network reconfiguration. During this re-estimation process, the VENTURE algorithm will not be available for execution.

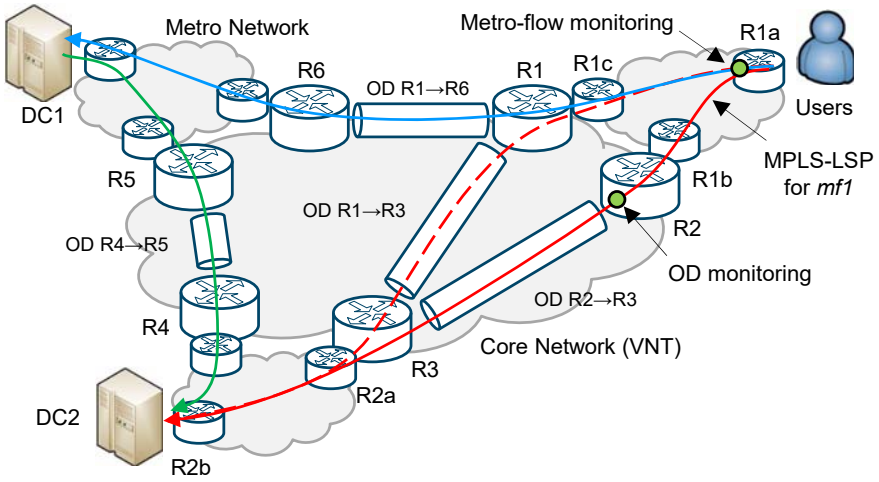

Fig. 8-1. OD traffic can change due to metro-wide re-configuration.

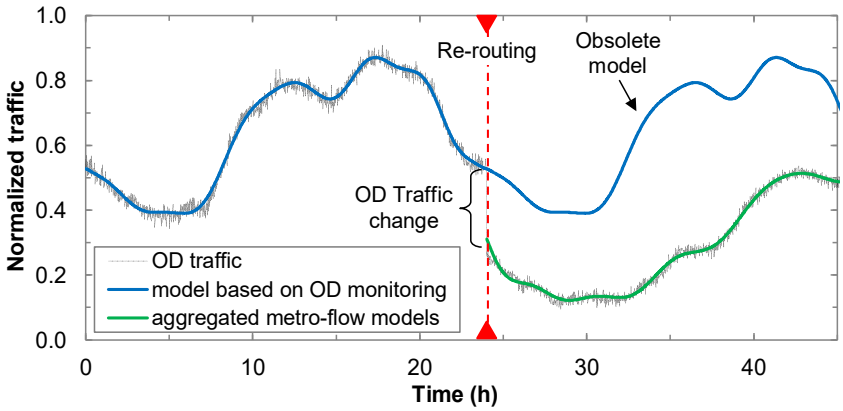

Fig. 8-2. Example of a traffic model that becomes obsolete.

Alternatively, OD traffic can be predicted by considering its relationship with metro-flow traffic. Effectively, by aggregating the traffic models of those metro-flows being routed through each core OD pair, we can produce new, updated OD traffic models. Immediately after the rerouting event, the obsolete model for ODs R1→R3 and R2→R3 are replaced by new ones based on the aggregation of metro-flows predictive models. Fig. 8-2 shows how the aggregation of metro-flow traffic predictions perfectly fits the new OD traffic pattern without the need of restarting the process of monitoring and estimation from scratch. By following this approach, the core network operator can keep the predictive capabilities, provided that some coordination between metro and core segments exist.

8.2 Metro-flow based OD pair traffic modelling

In Chapter 6, predictive models for the maximum OD bitrate were estimated aiming at re-configuring the VNT minimising traffic losses. Extending that approach to this new traffic scenario, predictive models for the maximum OD bitrate could be obtained by aggregating those for the maximum metro-flow bitrate. However, the aggregation of maximum predictors entails large overestimation as we prove next.

Let us consider a metro-flow f and the set Y_f containing all the data collected from f at the same period, but across m consecutive periods. Given a core OD pair od, we can consider the previous sets for all its metro-flows $F(od)=\{f_1,\ldots f_n\}$ and for itself, denoting it as Y_{OD}. It can be mathematically proven that the maximum value attained in Y_{OD} is upper bounded by the sum of the maximum values attained in each Y_f, as expressed in eq. (8-1). In other words, that the sum of the maximum metro-flow bitrate always overestimates the maximum OD bitrate. As a result, solving the VENTURE problem based on large traffic overestimation would entail overprovisioning, thus removing the efficiency of the algorithm. Therefore, we need to devise a traffic model aggregation procedure that allows predicting the maximum OD bitrate accurately. The magnitude of the overestimation above is numerically evaluated in Section 8.3.

$$\max(Y_{od}) \leq \sum_{f \in F(od)} \max(Y_f) \tag{8-1}$$

Instead, for a particular core OD pair od let us assume that predictive models for the mean (μ_f) and the variance (σ^2_f) are available for each metro-flow f in $F(od)$; the estimation of models for the mean and the variance is presented in detail in Chapter 5. From the linearity of the expectation [Ho13], the average OD traffic (μ_{od}) is equivalent to the sum of the metro-flow average traffic (eq. (8-2)). Regarding the OD pair variance (σ^2_{od}), it can be expressed as the summation of metro-flow variances if and only if variances are uncorrelated (eq. (8-3)). Correlation is commonly observed in the traffic and has been already studied in the literature

[Zha13]. Therefore, it would not be realistic to assume that the aggregated metro-flows have uncorrelated traffic if, for instance, they convey similar service traffic. When correlation is present between metro-flows, the expression of the OD variance becomes more complex since it additional nonzero covariances between all pairs of aggregated flows needs to be added [Ho13]. In Section 8.3 we analyse the bias introduced in the estimation of σ^2_{od} when covariance terms are excluded (linear aggregation).

Table 8-1 presents the proposed algorithm to create or update core OD traffic models after a metro-flow rerouting event. It receives the set Q with all OD pairs, where each pair includes its model m and the set of aggregated metro-flows. First, the set of obsolete models is found by inspecting the current aggregation of the metro-flows (line 1 in Table 8-1). For each obsolete OD model, the type of the model determines whether it is a model estimated from core traffic monitoring (NEW_CORE) (lines 4-5), from metro traffic monitoring (NEW_METRO) (lines 6-7) or it is a model that needs to be updated (UPDATE) by including the new metro-flows entering the OD pair and excluding those ones leaving it from the prediction (lines 8-9). For the updating process, we can take advantage of the linearity of the mean and the variance in the aggregation to produce updates applying equations (8-4) and (8-5), only taking into account those metro-flows leaving and entering the core OD.

Finally, the algorithm returns the set of updated models (line 9). Note that the aggregation of μ and σ^2 models entails adding the metro-flows piece-wise linear functions. However, this is immediate from the piece-wise linearity of these functions by simply adding the slopes and intercepts for each segment to obtain the aggregated piece-wise linear function for μ_{od} and σ^2_{od}. For the sake of simplicity, we assume that all aggregated models present the same period and number of segments. Otherwise, additional computation would be required to obtain the partitioning resulting from merging all the piecewise linear functions μ_f and σ^2_f using the least common multiple period.

$$\mu_{od}(t) = \sum_{f \in F(od)} \mu_f(t) \tag{8-2}$$

$$\sigma^2_{od}(t) = \sum_{f \in F(od)} \sigma^2_f(t) \tag{8-3}$$

$$\mu_{od}(t) += \sum_{f \in F_{IN}(od)} \mu_f(t) - \sum_{f \in F_{OUT}(od)} \mu_f(t) \tag{8-4}$$

$$\sigma^2_{od}(t) += \sum_{f \in F_{IN}(od)} \sigma^2_f(t) - \sum_{F_{OUT}(od)} \sigma^2_f(t) \tag{8-5}$$

Table 8-1: OD model update algorithm

```
INPUT Q = {<od, m, F(od)>}
OUTPUT S = {<od, m'>}
1:  Q_obs ← getObsoleteModels(Q)
2:  S ← ∅
3:  for each q = <od, m, F(od)> in Q_obs do
4:    if type(m) = NEW_METRO then
5:      m' ← m
6:    else if type(m) = NEW_METRO then
7:      m' ← newAggregate(F(od)) (eq. (8-2),(8-3))
8:    else if type(m) = UPDATE then
9:      m' ← updateAggregate(m, F(od)) (eq. (8-4),(8-5))
10   S ← S U {<od, m'>}
 :  return S
11
 :
```

Let us now analyse the worst-case time complexity of the previous algorithm, assuming that all OD models need to be re-estimated ($|Q_{obs}|=|Q|$) with a maximum number of metro-flows $|F|$ for each re-estimation. Let us also assume that all aggregated models are of type NEW_CORE (i.e., built from scratch using eqs. (8-2) and (8-3)) being this the most time-consuming case. Then, the worst-time complexity is O($|Q| \cdot |F| \cdot nSegm$).

Limiting the piece-wise linear model evaluation to the mean and the variance as presented in Chapter 5 discards other interesting estimations such as the maximum bitrate, important to re-optimise the VNT as presented in Chapter 6. Although the algorithm does not directly provide this estimation, we can obtain it in a later stage by applying results from probability theory involving μ and σ^2. Given a time t, let us assume that the traffic is distributed following a normal distribution $N(\mu, \sigma^2)$. Then, eq. (8-6) predicts the maximum bitrate with a confidence of 95% and 99.7%, respectively for k=2 and k=3 [Gr06].

$$\max_{od}(t) \approx \mu_{od}(t) + k\sqrt{\sigma^2_{od}(t)} \tag{8-6}$$

Finally, note that the previous prediction provides the maximum traffic with granularity T (e.g., 15 min), which might not be enough for VNT reconfiguration actions typically requiring the maximum predicted bitrate during larger intervals (e.g., one hour as proposed in Chapter 6). One solution to obtain predictions for larger intervals is to produce several predictions along the considered interval and keep the maximum value obtained. Although this procedure entails multiple evaluations of the model thus, increasing the complexity of the proposed algorithm, the number of these evaluations needed to ensure the highest prediction provided by the model is known. This follows from the fact that eq. (8-6) defines a piece-wise linear function for the maximum prediction and that the maximum value in a piece-wise linear function segment that takes place at the edges. Thus, it is enough

to produce predictions at those time points where two consecutive segments of the piece-wise linear functions are connected.

Note that the presented approach assumes that OD traffic can be accurately approximated as the sum of the metro-flow bitrate participating in the OD. In the next section, this assumption is validated for a wide range of traffic conditions.

8.3 Illustrative numerical results

In this section, we first evaluate the proposed approach to obtain OD core traffic models based on the aggregation of metro-flow traffic models. To that end, we analyse the key aspects regarding metro-flow model aggregation into core OD models and their use for VNT reconfiguration based on traffic prediction. Simulated metro and core-flow monitoring data for the next study was obtained following the same set up presented in Section 5.4 of this thesis.

8.3.1 Metro-flow model aggregation analysis

Let us start this study with a preliminary analysis validating the upper bound stated in eq. (8-1). This is, to assess that the sum of maximum predicted metro-flow bitrate overestimates the maximum OD bitrate. To that end, we ran simulations where an increasing number of metro-flows (Users and Datacenter) are aggregated into a single OD pair and monitored for 3 months. These traffic profiles correspond to those already detailed in Section 5.4.1. After this time, predictive models for the maximum bitrate of each metro-flow are estimated and the sum of their outputs compared with the maximum predicted OD bitrate, therefore evaluating the magnitude of the bound stated by eq. (8-1).

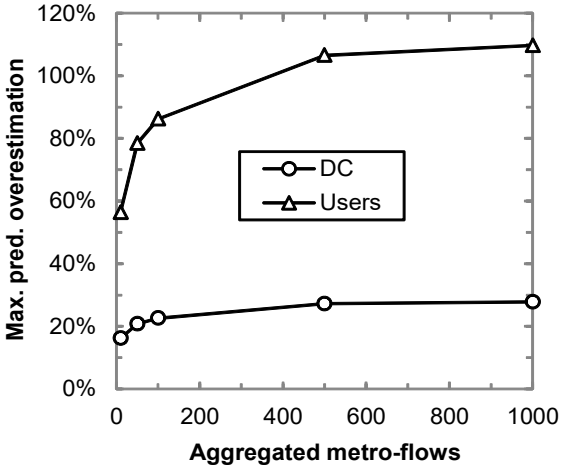

Fig. 8-3 Predictive error resulting from the aggregation of maximums.

Results in Fig. 8-3 show the relative overestimation that is introduced when the sum of maximum metro-flow bitrate is used as an estimator of the maximum OD bitrate. We can observe how this overestimation doubles the target prediction value when it is used under bursty traffic patterns such as Users. In contrast, for less variable traffic profiles such as Datacenter, the overestimation remains around 20%. In any case, the results validate eq. (8-1) and thus the impossibility of solving the VENTURE problem efficiently based on this prediction.

Next, we analyse under which traffic conditions the proposed predictive model aggregation (formally stated by eqs. (8-2) and (8-3)) is valid. These equations assume that the bitrate of an OD can be accurately approximated as the sum of the aggregated metro-flow bitrates. Therefore, the first analysis focuses on the traffic conditions that allow such approximation.

The scenario where several metro-flows are aggregated into a single core OD pair can be modelled using queuing theory; metro-flow packets arrive to a packet-switching node, where they are queued and aggregated into a single OD pair at a maximum give rate, e.g., 100 Gb/s. As proposed in [Gro08], this can be mathematically modelled by a G/D/1 queue, assuming a generic distribution of arriving packets and a single server with constant service time. In such queue, the sum of the metro-flow bitrates accurately approximates the resulting OD pair bitrate as long as the actual service rate does not exceed 90% of its maximum (queue length remains small) [Gro08]. Such condition must be ensured during the process of VNT reconfiguration, and therefore vlink capacities must be dimensioned to ensure that OD capacity utilisation is under the threshold above.

Let us finally evaluate the bias introduced in the estimation of σ^2_{od} (as presented in section 8.2) by running experiments where the maximum bitrate (eq. (8-6)) is predicted for a single OD pair aggregating Users metro-flows only, which leads to a large and positive covariance. Fig. 8-4a shows the minimum value of k needed to predict the maximum bitrate below a given error, for a different number of aggregated metro-flows. A value of k close to 6 suffices to ensure an error below 2% for any number of aggregated metro-flows. On the contrary, we observe that the value of k cannot be bounded to ensure a prediction error close to 0%; in fact, the minimum value of k seems to increase with the number of aggregated metro-flows. Finally, we observe that a value of approximately k=5.75 tightly bounds the prediction error below 1.6% for any number of aggregated metro-flows. For the particular case of 500 aggregated metro-flows, Fig. 8-4b illustrates the prediction error for as a function of k.

Finally, Fig. 8-5 illustrates the bitrate of an OD pair along the day mixing different metro-flow traffic profiles, as well as the predictions based on the proposed metro-flow model aggregation. Note that min and max models have been obtained for k=5. As expected, because of traffic aggregation, the variability of OD traffic is much smaller than that of metro-flows.

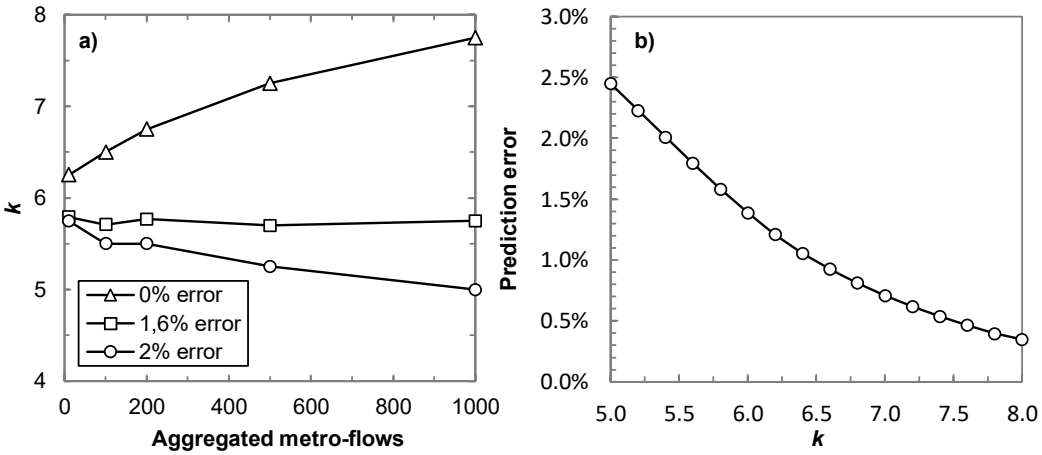

Fig. 8-4 (a) Value of k (eq. (8-6)) vs number of aggregated metro-flows and (b) prediction error vs value of k for 500 aggregated metro-flows.

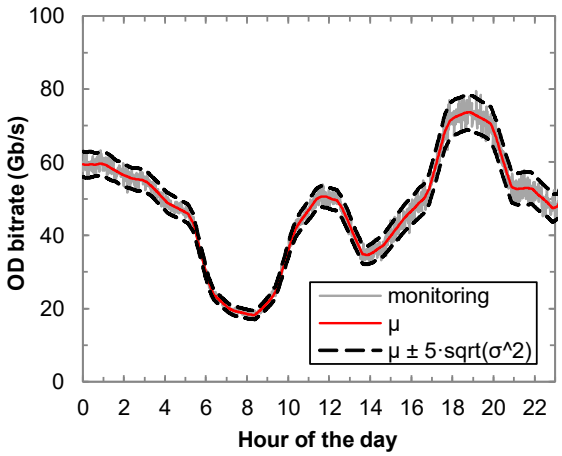

Fig. 8-5 Prediction of min/max/avg OD bitrate during one day.

8.3.2 VNT reconfiguration performance

For evaluation purposes, we ran simulations in iONESim considering a 14-node core VNT interconnecting seven metro networks using dual-homing, where 100 Gb/s lightpaths support vlinks in the metro areas. A total of 1,400 metro-flows following the Users and DC profiles presented in Chapter 5 were injected into the core VNT.

Once models have been estimated, two metro-flow rerouting actions are triggered daily in all metro areas, being the core controller notified for such changes. The first rerouting action, taking place at 7 am, splits the set of metro-flows entering the core VNT into two sets, each rerouted to one of the egress routers of the metro area in the hope of avoiding congestion. Two rerouting schemes for this action are considered:

- Randomized rerouting*:* the set of metro-flows is evenly split into two separate subsets at random, each rerouted to one egress router.
- Per type of service rerouting: the subset of all flows belonging to a randomly selected type of service is selected and rerouted towards the second router.

The second rerouting action takes place at 8 pm and groups all metro-flows to use one single egress router. Under these scenarios, we evaluated two different approaches to reconfigure the core VNT:

- The first approach, named as *threshold-based* uses a fully meshed VNT and increases or decreases the vlink capacity according to a capacity usage threshold. This approach is followed when the core controller is not able to rebuild obsoleted OD traffic models under frequent metro-flow rerouting.
- In the second approach, the VENTURE algorithm runs periodically, e.g., every hour, to optimise the VNT using updated OD traffic predictions based on metro-flow model aggregation.

Fig. 8-6 shows the maximum number of 100Gb/s transponders needed to convey core traffic under the randomised (a) and the per type of service (b) rerouting schemes, for a range of increasing traffic loads. Note that since the number of transponders is not limited in the nodes, both reconfiguration approaches offer zero blocking probability. It can be observed that VENTURE can adapt the VNT using fewer transponders than that of the threshold-based approach in the studied range of loads, producing savings up to 30% under the randomised rerouting scheme and up to 40% when per type of service rerouting is considered.

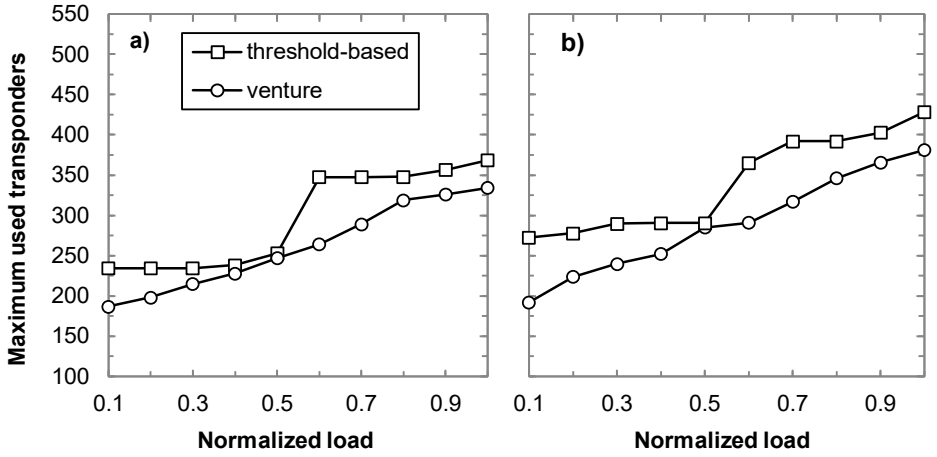

Fig. 8-6 Maximum used transponders under *randomised* (a) and *per type of service* (b) rerouting schemes.

8.4 Conclusions

Aggregated metro-flow traffic predictive model is proposed to cope with OD traffic changes in the core as a result of uncoordinated metro-flow rerouting, where the predictive traffic models are used to reconfigure the core VNT. By conveniently aggregating metro-flows after they become rerouted, OD predictive models can be rebuilt fast to keep the predictive capabilities in the core. To obtain quality metro-flow predictive models, an estimation algorithm that allows obtaining models that can be aggregated and evaluated efficiently is presented.

The process of metro-flow modelling was analysed, concluding that at least 2 months of monitoring data aggregated every 15 minutes are needed to obtain quality predictions for different traffic profiles. The aggregated traffic model was then used as input for a VNT reconfiguration algorithm, which was evaluated against a threshold-based approach used when new models cannot be rebuilt under changing OD traffic, obtaining savings as high as 40% regarding used transponders.

Chapter 9

Experimental validation

This chapter presents the network architectures that allow bringing the procedures and algorithms from the previous chapters to experimental environments. Two different and complementary network control and management architectures are motivated, presented and experimentally assessed.

The first architecture aims at enabling traffic monitoring and data analytics with predictive models in the network controller. The architecture includes a central domain controller performing data analytics and is based on the extended node presented in [Gi17]. To demonstrate the proposed architecture, an OAA loop use case is defined along with their respective workflows. The use case contains *i*) predictive traffic model estimation and *ii*) periodic VNT reconfiguration based on traffic prediction, both supported by the algorithms devised in Chapter 5 and Chapter 6 for traffic modelling and for solving the VENTURE problem, respectively.

The second architecture complements the previous by enabling the sharing of predictive models across multiple data analytics -based network controllers. This architecture features a flow controller, an entity responsible for storing and disseminating updated traffic models across the participating controllers. To demonstrate this architecture, one use case is presented along with three workflows demonstrating the storage and dissemination of updated predictive traffic models between metro and core controllers. This use case paves the way to the aggregation of metro-flow traffic models into core-flow ones in experimental environments, based on the ideas presented in Chapter 8.

Both architectures are experimentally assessed by implementing the aforementioned use cases in a distributed test-bed connecting premises from UPC (Barcelona, Spain) and Scuola Sant'Anna/CNIT (Pisa, Italy).

9.1 Bringing data analytics to the network

In this section we motivate the proposed architecture, whose main concept is illustrated in Fig. 9-1, where optical and MPLS packet nodes that are extended with monitoring capabilities [Gi17] provide monitoring data to a centralised control and management system (hereafter *domain controller*) extended with data analytics capabilities.

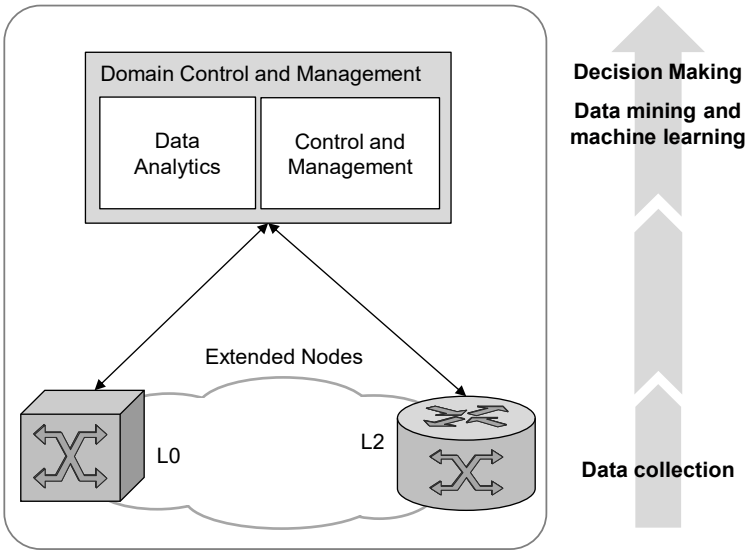

Fig. 9-1. Conceptual architecture.

In connection-oriented networks, typical *observation points* are: *i*) optical connections, where bit error rate (BER) and Tx/Rx optical power, among other parameters, can be measured at the optical transponders; and *ii*) MPLS LSPs, where packets and bit rate can be measured in any MPLS switch along the route of the LSP.

Although extended nodes enable monitoring for data analytics, the scarcity of computational and storage resources becomes insufficient when large datasets need to be processed and global network knowledge to be discovered. Consequently, data analytics capabilities are needed at the domain controller. The domain data analytics is thus responsible for: *i*) collating, processing, and storing monitoring data and *ii*) managing the configuration of extended nodes. With this global vision, the domain analytics is able to perform a network-wide knowledge discovery in data (KDD). Since large amounts of data need to be collected, stored and analyzed, the domain data analytics module needs to be designed with big data capabilities.

Aiming at facilitating decision-making, we extend the definitions in [RFC7011] and propose a monitoring hierarchy consisting of three different elements: *i*) *observation points* are locations in the network where monitoring data records are

generated (as in [RFC7011]), *ii*) *observation groups* are sets of observation points, which monitoring data needs to be aggregated. An observation group can aggregate monitoring data from observation points defined in different nodes; and *iii*) *monitoring entities*, as sets of observation groups, that gather monitoring data collected from related observation groups, possibly in different layers.

For illustrative purposes, Fig. 9-2 presents examples of the proposed monitoring hierarchy. Fig. 9-2a shows three separate LSPs that convey the traffic of OD pair R1→R3. To measure the OD traffic, that of each LSP needs to be metered and aggregated. Packet LSPs can be monitored by activating one single observation point in any packet node along their route, which is advantageous since the number of observation points that can be configured in every packet node is limited, as already introduced. Notwithstanding observation points have been activated for the three LSPs in our example in Fig. 9-2a, not all of them are in the same packet node, so an *observation group* that aggregates monitoring data records received from every LSP supporting the OD traffic is configured.

This hierarchy is set up in the domain controller and sent to the corresponding nodes so that data can be aggregated, if possible, directly in the extended node or, otherwise, in the domain controller. In the example, node R1 collects traffic samples for LSP-1 and LSP-2 and exports them to the domain controller as a single, aggregated traffic sample for OD R1→R3. The remaining observation point data for LSP-3 in node R2 will be aggregated to that from node R1 once data arrive in the domain controller, to produce the final sample for the OD.

Observation groups aggregate data of the same type. To relate observation groups from related connections collected at different network layers belonging to the same network entity, we define a *monitoring entity*, which is identified by a *symbolic name*. To illustrate this, Fig. 9-2b shows an MPLS VNT where the capacity of the virtual link (*vlink*) R1→R2 is supported by a lightpath in the optical layer. Observation group 1 is configured to monitor the bitrate conveyed through the virtual link, whereas observation group 2 monitors the optical path BER. By defining a *virtual link* monitoring entity parenting both observation groups, heterogeneous data correlation across multiple network layers becomes easier.

Finally, a custom monitoring templating system based on the IPFIX protocol [RFC7011] is included to support monitored data pre-processing and transformation in the extended nodes. Templates allow defining custom fields for collected and exported data.

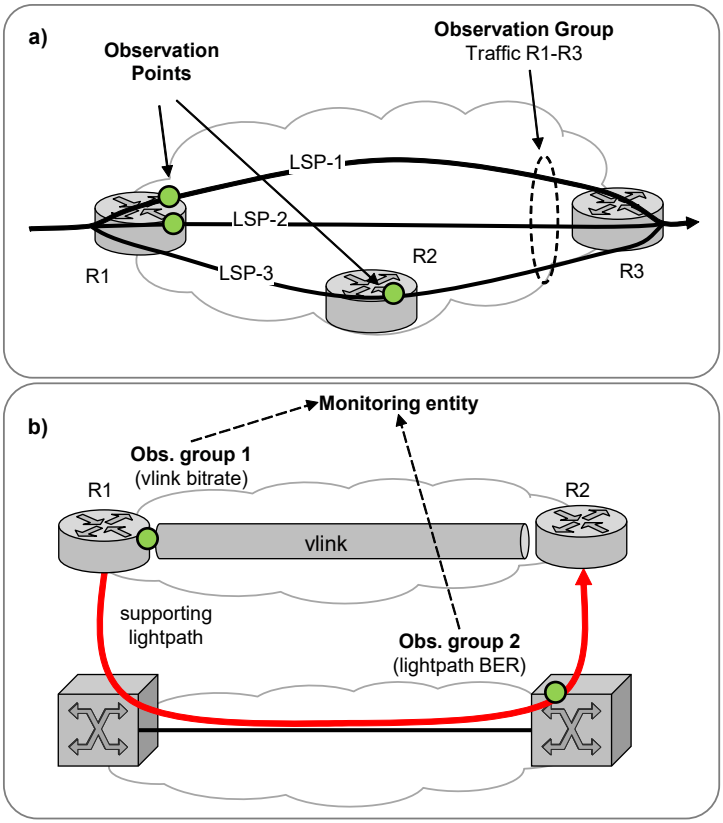

Fig. 9-2. Proposed monitoring hierarchy.

9.2 Proposed architectures

9.2.1 Domain controller architecture

Fig. 9-3 presents the proposed architecture, where extended nodes [Gi17] receive IPFIX protocol messages from physical devices containing data records from multiple observation points. To configure the extended node, the controller pushes a set of observation groups, where each of them defines a set of rules used to map data records from specific observation points to that observation group. When a new IPFIX message arrives at the extended node, the IPFIX speaker creates a list of pairs key-value (sample) for every data record in the message; in addition to header fields such as TimeStamp, data records contain a set of fields which structure (template) is identified by the field DataSet Id.

Monitored data exported from the extended nodes is collected at the domain controller by an IPFIX speaker that forwards it to a data manager module in charge of processing samples and storing them into a collected distributed big data repository (in our implementation we use Cassandra [Cas]). To adequately handle

samples, observation group handlers are defined to ensure the right aggregation of data coming from observation points in different nodes, finally generating notifications upon the reception of new samples.

To enable data analytics and KDD implementation in the domain controller, a KDD module is defined; it contains KDD processes that execute intensive data analytics tasks, either locally or delegated to a computing cluster running, e.g., Spark [Spark]. These processes can be managed and configured by an NMS from an external configuration database in the domain controller. A KDD API is included to facilitate the implementation of new KDD processes; it allows accessing and storing data in the collected repository, accessing the process configuration database, and sending commands to the domain controller.

The analytics controller triggers the execution of KDD processes. KDD processes need to subscribe to the notifications they can receive; upon the reception of a notification in the KDD module, all KDD processes subscribed to such type of notification are run.

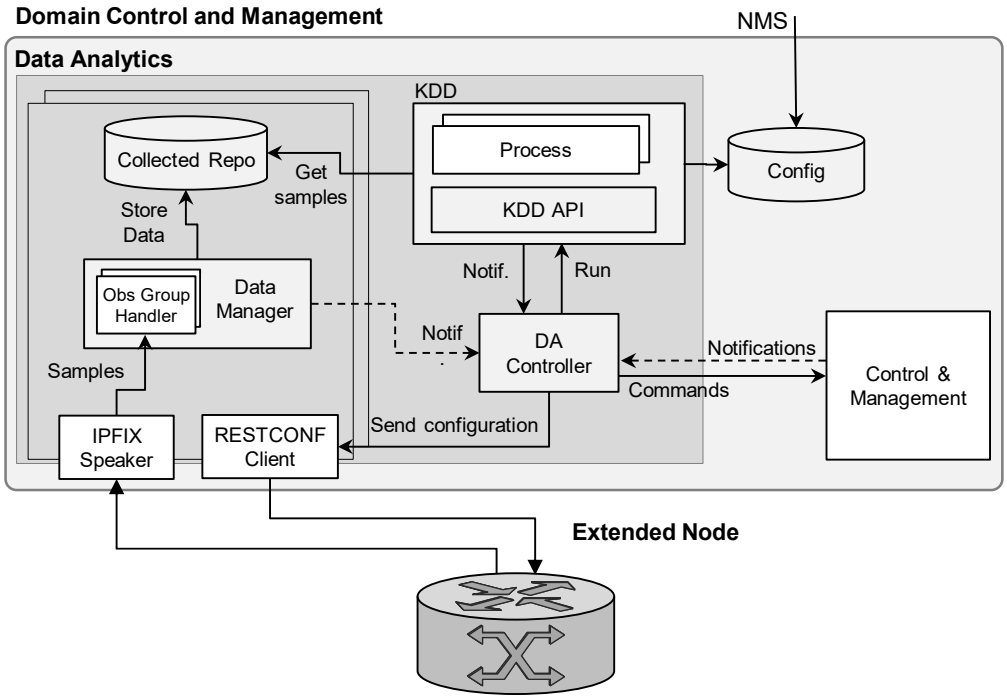

Fig. 9-3. Detailed architecture, using the extended node proposed in [Gi17].

Besides the domain analytics module, the control and management system is in charge of the network control and management. The domain analytics controller can send commands to trigger actions in the network using a REST API; this interconnection between the domain analytics and the control and management system is key to enable implementing the OAA loop.

9.2.2 Multi-domain controller architecture

Although the previous architecture allows to monitor and obtain predictive traffic models, its scope is limited to the observed network domain, thus lacking a mechanism to obtain data analytics -based knowledge from foreign network domains. This mechanism is key in order to enable the aggregation of metro-flow predictive models into core-flow predictive models as presented in Chapter 8 aiming at ensuring uninterrupted prediction. To alleviate this problem and complete the previous architecture, we propose a complementary control architecture that enables the sharing of predictive among multiple domain controllers.

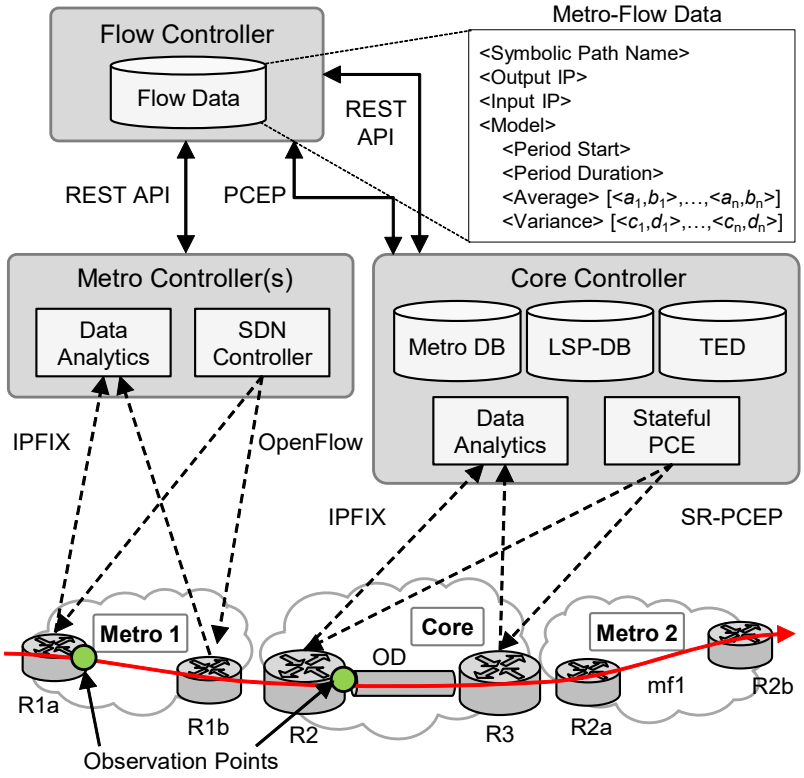

Fig. 9-4. Proposed flow controller architecture.

Fig. 9-4 presents the proposed architecture where every metro controller contains a data analytics module based on the previous architecture, capable of storing and processing metro-flow monitored data to estimate metro-flow predictive models. The metro controller also includes an SDN controller [SDN] responsible for the configuration of the network devices. The core controller contains an analogous analytics module for OD traffic monitoring and prediction and includes an additional database to store the metro-flow -related data. A Traffic Engineering

Database (TED), a Label Switched Path Database (LSP-DB), and a Stateful Path Computation Element (PCE) complete the core controller's architecture [RFC7491].

We propose a centralised flow controller with a repository to store flow-related data that is updated from the metro controllers and used by the core controller to produce predictive OD traffic models. The repository stores for each metro flow: *i*) its LSP's symbolic path name; *ii*) the border metro nodes through which the flow leaves and enters different metro areas and *iii*) the metro-flow predictive model. Since periodic models are assumed, the model includes its period and starting time, as well as two piecewise linear functions defining the average bitrate and its variance along the period.

Next section presents the use cases and the corresponding workflows that are executed on the two architectures above, to apply the OAA loop for VNT reconfiguration based on traffic prediction and to share updated predictive models.

9.3 Use cases and workflows

9.3.1 Traffic monitoring and model estimation

In the first workflow (Fig. 9-5) monitoring data is collected, and a traffic model for an LSP (LSP 02-05 in Fig. 9-7) is estimated. MPLS nodes send IPFIX messages encoding data records with monitored traffic data to the extended node (message 1 in Fig. 9-5).

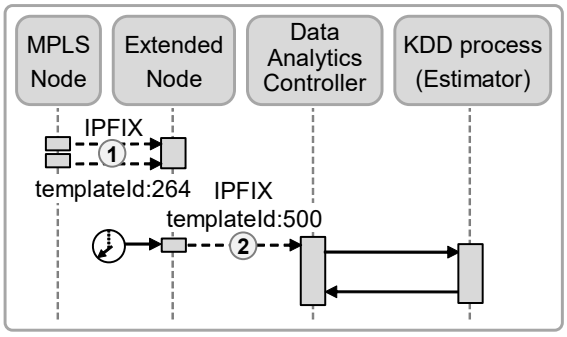

Fig. 9-5. Workflow for traffic monitoring and model estimation.

Open Virtual Switches (OVS) can use template ID 264 [Tr13] to that end whose details are shown in Table 9-1. The monitored samples are aggregated, stored and periodically sent to the domain controller according to some period parameter *T*. The extended nodes export the aggregated samples using a possibly different template. In this case, we defined template ID 500 with the structure defined in Table 9-2. Note that in addition to traffic counters, the symbolic name and the observation group Id are defined in template Id 500.

Table 9-1: IPFIX template 264 (OVS native)

Field Name	Description
observationPointId	Id of the observation point.
flowDirection	Direction of the flow.
src/dst MacAddress	IEEE 802 source and destination MAC addreses.
ethernetType	Ethernet type that identifies the client protocol in the payload.
ethernetHeaderLength	Ethernet header length.
flowStart/StopDeltaMicros	Time offset relative to the message timestamp for the first/last observed packet of this flow.
packetDeltaCount	Number of packets since the previous report.
layer2OctetDeltaCount	Number of L2 octets since the previous report.
flowEndReason	The reason for terminating the flow and sending this msg.

Table 9-2: IPFIX template 500 (custom definition)

Field Name	Description
symbolicName	String with the Symbolic Name of the monitored flow.
observationGroupId	Id of the observation group.
flowDirection	Direction of the flow.
packetDeltaCount	Number of packets since the previous report.
layer2BitDeltaCount	Number of bits since the previous report.
flowDurationMilliseconds	Time between the first and last observed packets of the flow.

When a new sample arrives in the domain controller, the data manager stores it in the collected data repository and notifies the data analytics controller, which forwards the notification to the KDD module that will select the KDD process subscribed to such notifications; in this case, the traffic estimator. The estimator KDD process will eventually produce a new predictive traffic model that it stored in the configuration.

The KDD estimator process algorithm is presented in Table 9-3 and is executed every time a new traffic sample arrives in the domain controller (e.g., every 15 min). It receives as input an observation group Id, the time window within which

traffic samples will be considered for model estimation, the minimum number of samples to estimate the model, and the last observed errors between the model and sampled traffic.

The algorithm starts by retrieving the configuration of the observation group from a configuration database, the current predictive model if it exists, and the last received traffic sample (lines 1-3): The configuration contains parameters like the monitoring period (e.g., 15 min). In case that a predictive model is currently active, it is re-estimated if its predictive quality drops reflecting an increasing series of errors (lines 4-7). Otherwise, the estimator returns (line 8).

To estimate a predictive model, a query filter is prepared to retrieve monitored samples in the specified time window (line 9). The number of available traffic samples within the time window are first counted (line 10), and if there are enough to produce a new model, the previous filter is used to query the collected data repository (lines 11-12). The resulting dataset is used as input of the mathematical procedure that computes the predictive model, presented in Table 5-2 of Chapter 5. Once the model has been estimated, it can be evaluated to obtain a bitrate prediction following the evaluation procedure presented in Table 5-3.

Table 9-3: KDD estimator algorithm

```
   INPUT obsGroupId, window, minSamples, errors
 1:  config ← getConfig(obsGroupId)
 2:  model ← getModel(obsGroupId)
 3:  z = <y, time> ← getLastSample(obsGroupId)
 4:  if model ≠ Ø then
 5:     error ← computeError(model, z)
 6:     errors[obsGroupId].append(<z.time, error>)
 7:     if NOT needsEstimation(errors[obsGroupId]) then
 8:       return
 9:  filters ← (t_begin = z.time – window, t_end = z.time)
10:  count ← sampleCount(obsGroupId, filters)
11:  if count < minSamples then return
12:  Z ← getSamples(obsGroupId, filters)
13:  model ← estimatePiecewiseLinearModel(Z, config)
14:  updateModel(obsGroupId, model)
15:  sendNotif(obsGroupId, MODEL_CHANGED)
16:  return
```

9.3.2 VNT reconfiguration based on traffic prediction

The second workflow (Fig. 9-6) focuses only on monitoring and data analytics, whereas the messages inside the controller (e.g., ABNO [RFC7491]) and between the controller and the MPLS nodes are not shown for the sake of clarity. A VNT traffic predictor process in the domain analytics KDD is periodically executed to predict a traffic matrix used as input to reconfigure the VNT. This prediction is

made using the predictive models produced as explained before. For illustrative purposes, Fig. 9-7 shows an example of VNT reconfiguration, where predictive traffic models have been estimated in Fig. 9-7a. According to the traffic estimated for the next period, LSP 02-05 is rerouted, and vlink R3-R4 is torn down in Fig. 9-7b.

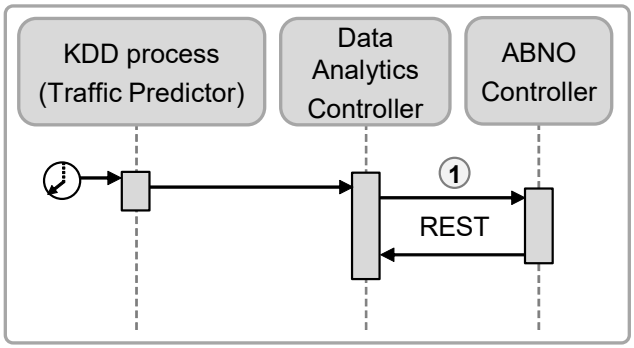

Fig. 9-6. Workflow for VNT reconfiguration based on traffic prediction.

At regular intervals (e.g., every hour), the VNT traffic prediction process generates a new traffic matrix for the OD traffic in the VNT for the next period. The predicted matrix along with a VNT reconfiguration request is sent to the data analytics controller, which notifies to the ABNO controller using a REST request (message 1 in Fig. 9-6). The ABNO controller follows an internal workflow and eventually sends the appropriate commands to the MPLS.

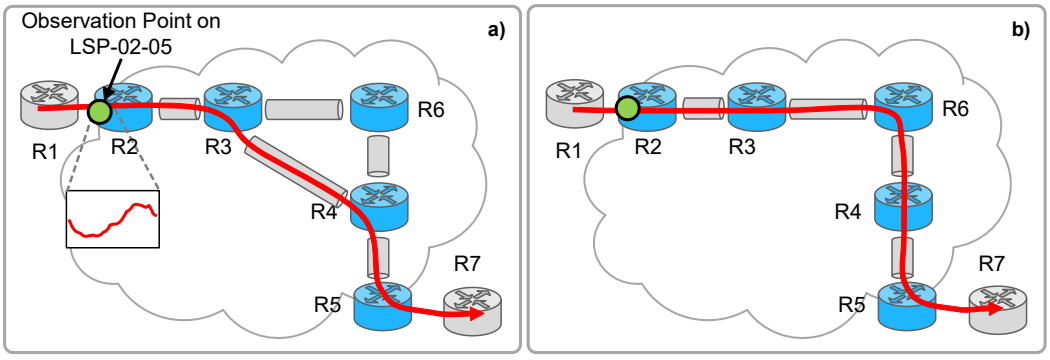

Fig. 9-7. Example of VNT reconfiguration based on traffic prediction.

9.3.3 Metro-core model dissemination

Three workflows are defined to store and retrieve metro-flow data in/from the flow controller. Fig. 9-8a shows the first workflow triggered when a new LSP for a metro-flow is set-up across different metro domains. Every metro-controller involved in the LSP creation sends a JSON-encoded REST API message to the flow controller with the LSP symbolic path name and the input or output metro border

node. The flow controller creates a new entry in the flow repository and populates it correlating the data received from different metro controllers.

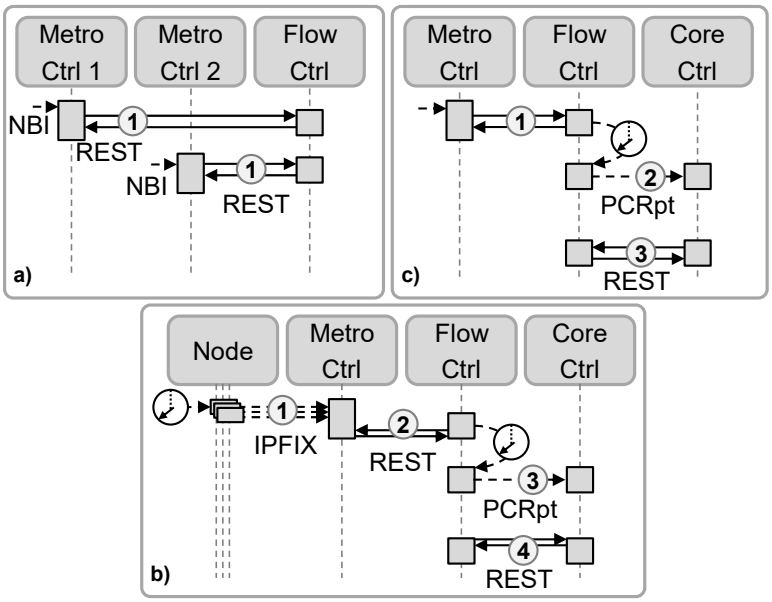

Fig. 9-8. Metro-flow set-up (a), Metro-flow model estimation (b) and Metro-flow re-routing (c).

Once the LSP for the metro-flow is operational, its traffic is monitored in any of the nodes in the source metro area and monitored data is exported by means of IPFIX messages (message 1 in Fig. 9-8b) to the metro controller. When enough monitoring data has been collected to estimate a new metro-flow model, or to re-estimate an existing one, the metro controller sends the predictive model to the flow controller in a REST API message, including the LSP's symbolic path name for identification (message 2).

After a metro-flow data entry has been completed or updated, the flow controller notifies that to the core controller by issuing a PCEP PCReport message [RFC8261] (message 3) containing the list of updated metro-flow LSPs. A delay has been introduced to prevent flooding the core controller with several updates from different metro controllers. The core controller can now retrieve updated metro-flow data by issuing a REST API request (message 4), being thus able to update obsolete OD traffic models. When a metro controller decides to re-route one or more metro-flows, it updates the metro-flow data in the flow controller (message 1 in Fig. 9-8c), which proceeds as described for Fig. 9-8b.

In next section, we experimentally assess the two proposed architectures and their respective workflows.

9.4 Experimental assessment

9.4.1 Traffic monitoring and model estimation

The experimental validation of the domain controller architecture was carried out on a distributed testbed connecting premises in Scuola Sant'Anna/CNIT (Pisa, Italy) and UPC (Barcelona, Spain), where the multi-domain MPLS-over-optical VNT depicted in Fig. 9-7 was considered. The Scuola Sant'Anna/CNIT domain includes 4 commercially available Juniper MPLS routers, and two Ericsson SPO 1400 ROADM employ 10/100 Gb/s transceivers. These network devices are controlled by a multi-layer domain SDN controller developed in C++. The UPC domain includes OVSs nodes running on a Mininet environment; these devices are controlled by a Ryu SDN controller [RyuSDN] running on a virtual Linux host. UPC's SYNERGY testbed also includes the extended nodes running on top of the OVSs and the data analytics -based domain controller all implemented in Python. The control and management system used in the domain controller followed the ABNO architecture and was implemented in C++ [Ve15]; it includes an ABNO controller, a virtual network topology manager (VNTM) and a provisioning manager (PM) that communicates with both SDN controllers.

An observation point is defined in router R2 to monitor LSP 02-05 traffic, which follows a daily traffic profile. Regarding monitoring configuration, parameter T is set to 15 min. The collected data repository and predictive models are initially empty, and the estimator algorithm is configured to require at least 20 days of traffic samples to produce accurate enough predictive models.

Fig. 9-9 shows the meaningful messages exchanged between node R2, the related extended node, and the domain analytics module; messages contents are also detailed in Fig. 9-9. For the sake of clarity, messages are identified following workflow in Fig. 9-5. Node R2 sends data records in IPFIX messages to the extended node (1) containing bitrate information. Every 15 minutes (i.e., 900 seconds) the extended node sends an IPFIX message to the domain controller (2) with aggregated traffic data for the observation group. Aggregated samples are stored in the domain controller and trigger the execution of the estimator algorithm, which does not produce a predictive model until enough monitoring data (i.e., 20 days) is available. Once the model is estimated, it is locally stored and sent to the extended node R2 (3) as a JSON encoded object (Fig. 9-10).

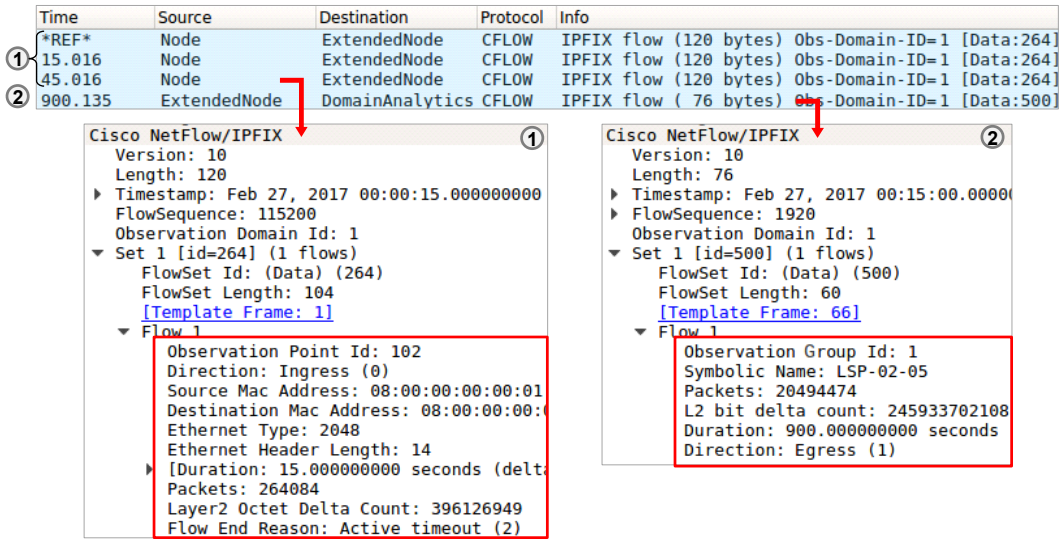

Fig. 9-9. Exchanged monitoring messages for the first workflow and details of the messages.

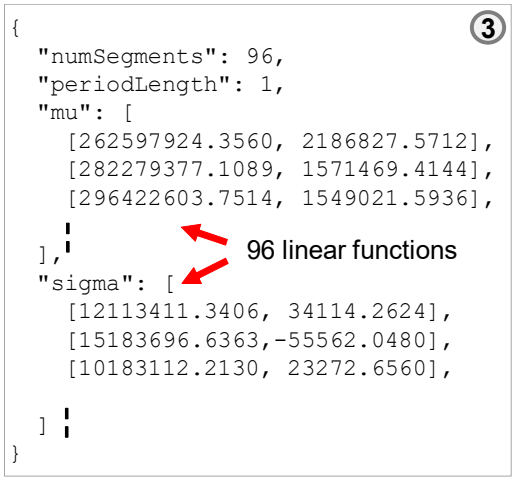

Fig. 9-10. JSON object storing the predictive model for LSP-02-05.

9.4.2 VNT reconfiguration based on traffic prediction

Let us imagine that predictive models are now available for every OD pair in the VNT; then, the VNT can be reconfigured based on traffic prediction at regular intervals (e.g., every hour). Fig. 9-11 lists the messages related to the workflow in Fig. 9-6 as well as messages between the ABNO components.

Time	Source	Destination	Protocol	Info
REF	(1) DomainAnalytics	ABNOCtrl	HTTP	POST /abno/vnt-reconfiguration HTTP/1.1
0.000	ABNOCtrl	VNTM	PCEP	Path Computation Request
0.097	VNTM	PM	PCEP	Path Computation LSP Update Request
0.138	PM	SDNCtrl	HTTP	PUT /cconn HTTP/1.1
5.989	SDNCtrl	PM	HTTP	HTTP/1.1 200 OK
5.989	PM	VNTM	PCEP	Path Computation LSP State Report
5.989	VNTM	PM	PCEP	Path Computation LSP Initiate
5.989	PM	SDNCtrl	HTTP	DELETE /cvlink HTTP/1.1
12.002	SDNCtrl	PM	HTTP	HTTP/1.1 200 OK
12.002	PM	VNTM	PCEP	Path Computation LSP State Report
12.002	VNTM	ABNOCtrl	PCEP	Path Computation Reply
12.002	ABNOCtrl	DomainAnalytics	HTTP	HTTP/1.1 200 OK

Fig. 9-11. Exchanged messages for VNT reconfiguration based on traffic prediction (Fig. 9-6).

The workflow is triggered when the domain analytics sends a REST request to the ABNO controller containing a VNT reconfiguration action together with the predicted traffic matrix (1). The ABNO controller identifies the action and delegates the request to the VNTM that solves the VENTURE problem and returns the new topology and the capacity of every vlink, together with the new route for every LSP.

As a result of the reconfiguration, LSP 02-05 is rerouted and vlink R3-R4 torn down, as illustrated in Fig. 9-7b. To implement this solution in the VNT, the VNTM first sends a PCUpdate message to the PM to communicate the LSP rerouting; the PM forwards the rerouting implementation to the corresponding domain SDN controller using a REST API request. Once LSP 02-05 has been successfully rerouted the VNTM receives a PCReport notification, and it sends a PCInit message to the PM to tear down vlink R3-R4; to that end, the PM sends a new REST API request to the SDN controller. Once the solution is successfully implemented in the network, the VNTM notifies it to the ABNO controller by sending a PCReply message, followed by the final reply from the ABNO controller to the initial VNT reconfiguration request by the domain analytics.

9.4.3 Metro-core model dissemination

The experimental validation of the flow controller architecture was carried out on a distributed test-bed connecting CNIT (Pisa, Italy) and UPC (Barcelona, Spain) premises. A core controller with segment-routing capabilities [Sga16] was located at CNIT, whereas the flow controller and two metro controllers were located at UPC. The core and metro controllers were extended with an HTTP REST API interface to exchange JSON-encoded messages with the flow controller. The SDN controller used in the metro areas is based on RyuSDN and uses OpenFlow to configure the network nodes. Finally, a set of extended nodes4 implemented on top of OpenVSwitches were deployed using Mininet at UPC premises to allow monitoring traffic.

Fig. 9-12a lists the REST API messages exchanged between the metro controllers and flow controller after an LSP for a metro-flow across the two metro domains is set-up. Messages specify the LSP's symbolic path name (LSP-01-02) and the IP address of the metro border node. Fig. 9-12b shows an IPFIX flow message (labelled as 1 in Fig. 9-8b) containing monitoring data from LSP-01-02 that is sent to the source metro controller for traffic model estimation. After collecting enough traffic data, a predictive model is estimated by the metro controller and sent to the flow controller in a JSON-encoded REST API message (message 2). The details are shown in Fig. 9-13a and include the LSP symbolic path name and the data representing the metro-flow predictive model.

Next, the flow controller issues a PCEP PCReport message to the core controller notifying the new data available for LSP-01-02 (message 3). The PCReport message contains a list of Stateful Request Parameters (SRP) and one LSP object with the LSP's symbolic path name. The core controller then issues a REST-API request with the symbolic path name of the LSP (message 4) to retrieve its data (Fig. 9-13b).

Finally, Fig. 9-12c lists the messages exchanged as a result of re-routing of LSP-01-02. First, the new border output node is sent by the metro controller in a REST API message to the flow controller (labelled as 1; in Fig. 9-8c). Once the flow controller updates the metro-flow data, equivalent messages to those for model creation are exchanged with the core controller to allow obtaining updated data for the re-routed LSPs (notice the updated border node in Fig. 9-13c).

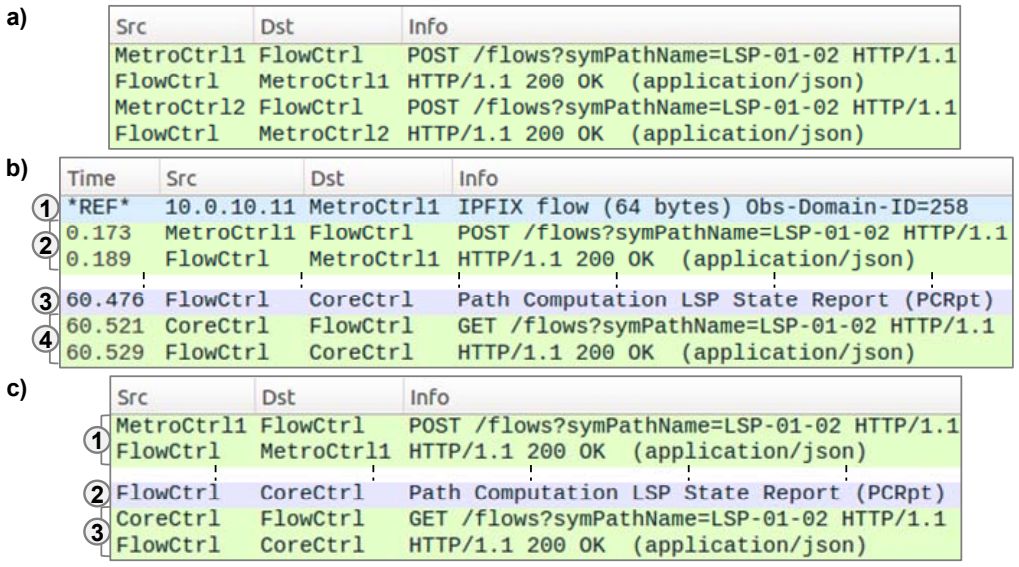

Fig. 9-12. Messages list for metro-flow set-up (a), metro-flow traffic model update (b) and metro-flow LSP re-routing (c).

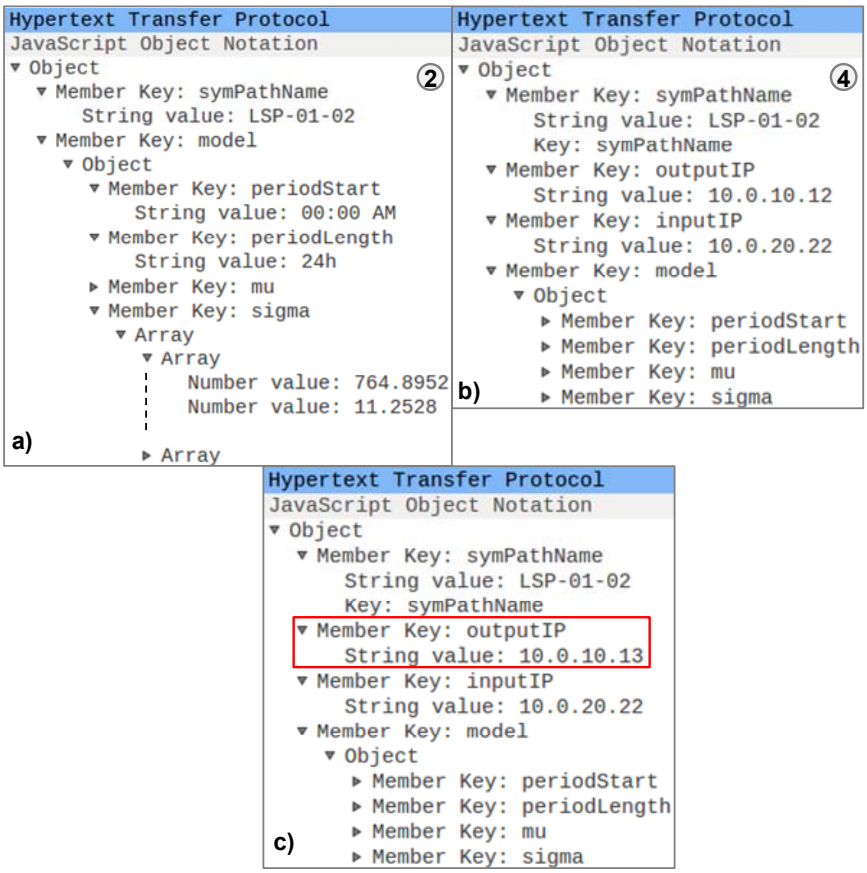

Fig. 9-13. Details of traffic model (a), metro-flow (b) and updated metro-flow (c).

9.5 Conclusions

The dynamicity introduced by new internet services requires from control and management architectures to help reducing resource overprovisioning. In this paper, we propose a new monitoring and data analytics network architecture where data analytics is applied at the domain controller.

In that regard, this paper proposed a hierarchical monitoring and data analytics architecture based on extended nodes that collate monitoring data and a centralised domain data analytics module that is able to perform network-wide data analytics on monitoring data collected from all the nodes in the network. The IPFIX protocol is used to define custom templates to convey monitoring samples from the different network layers. In that regard, a monitoring hierarchy that allows aggregation of samples from logically-related observation points, as well as to correlation of data from observation groups belonging to the same resource entity, was defined.

The proposed architecture was demonstrated through an OAA loop use case, and workflows on the proposed architecture were designed. First, a predictive L2 traffic model estimation and dissemination is proposed. The predictive model structure as well as the algorithms to estimate such models follow the ones presented in Chapter 5. An algorithm to estimate such models from monitored data was also presented. Next, predictive models were used to demonstrate periodical VNT reconfiguration based on traffic prediction.

To complement the previous architecture, a flow controller architecture is also proposed to allow adjacent domain controllers to share predictive traffic models among them. This facilitates the core controller re-estimating obsolete OD traffic models applying statistics for metro-flow model aggregation, as proposed in Chapter 8.

One use case composed by three workflows has been proposed to keep updated metro-flow data in the flow controller, either triggered by the estimation of a new predictive traffic model or for a re-routing action modifying the OD traffic aggregation in the core network. In any case, these actions originated by metro controllers are properly notified to the flow controller and eventually to the core controller.

The proposed architectures were experimentally assessed by implementing the proposed use cases on a distributed testbed connecting premises in UPC and Scuola Sant'Anna/CNIT.

Chapter 10

Closing discussion

10.1 Main Contributions

The main contributions of this thesis are:

- In Chapter 4, the iONESim network simulator was introduced as a means of evaluating the algorithms for cognitive in-operation network planning proposed in this thesis. The simulator allows combining in-operation network planning with data analytics based on monitoring data as well as the simulation of realistic metro and core flow bitrate based on a traffic generation framework.
- In Chapter 5, network traffic flow modelling was studied, and two different traffic modelling approaches proposed: a piece-wise linear model providing a constant-time, point-wise time evaluation and an ANN model capable of adapting to evolutionary traffic and requiring fewer assumptions about the underlying traffic distribution. The data analytics architecture needed to collect, process and transform monitoring data into both types of traffic models were proposed as well as the algorithms for their estimation and evaluation.
- In Chapter 6, The VENTURE problem was presented to reconfigure the VNT based on traffic prediction periodically. The benefits of using VENTURE against a reactive, threshold-based VNT reconfiguration were demonstrated by evaluating both approaches through simulation, obtaining savings between 8% and 42% regarding used transponders.
- In Chapter 7, advanced solving methods for VENTURE were studied to evaluate the extent of improvement of this new VNT reconfiguration

approach. Two state-of-the-art metaheuristics for combinatorial network optimisation (GRASP and ILS) as well as the original VENTURE algorithm were evaluated, obtaining an optimality gap between 5% and 9% within 1 minute of execution, being ILS the overperforming option.

- To ensure prediction under changing traffic patterns in the core, in Chapter 8 we proposed to extend traffic modelling to metro areas and aggregate the resulting metro-flow predictive models into quality OD core traffic models while in-operation. The problem of metro-flow model aggregation into quality core-flow models was mathematically studied and evaluated through simulation, showing the benefits of achieving uninterrupted quality prediction.
- Finally, in Chapter 9 we devised network architectures to bring the algorithms of the previous chapters to real environments. To enable the traffic modelling and VNT reconfiguration (Chapter 5 and Chapter 6) a domain control and management architecture was proposed. To support metro-flow model aggregation (Chapter 8) a flow controller was proposed. These architectures were experimentally assessed through a series of workflows in a distributed test-bed connecting UPC and CNIT premises.

In view of the above, we can conclude that the goals of this thesis have been successfully reached.

10.2 List of Publications

10.2.1 Publications in Journals

[Mo17-1] **F. Morales**, Ll. Gifre, F. Paolucci, M. Ruiz, F. Cugini, P. Castoldi, and L. Velasco, "Dynamic Core VNT Adaptability based on Predictive Metro-Flow Traffic Models," in Journal of Optical Communications and Networking, vol. 9, pp. 1202-1211, 2017.

[Mo17-2] **F. Morales**, M. Ruiz, Ll. Gifre, L. M. Contreras, V. López, and L. Velasco, "Virtual Network Topology Adaptability based on Data Analytics for Traffic Prediction," (Invited) in Journal of Optical Communications and Networking, vol. 9, pp. A35-A45, 2017.

10.2.2 Publications in Conferences

[Mo17-3] **F. Morales**, Ll. Gifre, F. Paolucci, M. Ruiz, F. Cugini, L. Velasco and P. Castoldi, "Experimental Assessment of a Flow Controller for Dynamic Metro-Core Predictive Traffic Models Estimation," in Proceedings of the European Conference on Optical Communication (ECOC), 2017.

[Mo17-5] **F. Morales**, M. Ruiz, and L. Velasco, "Data analytics based origin-destination core traffic modelling," in Proceedings of the International Conference on Transparent Optical Networks (ICTON), 2017.

[Mo17-6] **F. Morales**, M. Ruiz, and L. Velasco, "Core VNT Adaptation Based on the Aggregated Metro-Flow Traffic Model Prediction," in Proceedings of the Optical Fiber Communication Conference (OFC), 2017.

[AV16] A. P. Vela, A. Via, **F. Morales**, M. Ruiz, and L. Velasco, "Traffic generation for telecom cloud -based simulation," in Proceedings of the International Conference on Transparent Optical Networks (ICTON), 2016.

[Gi16-2] Ll. Gifre, **F. Morales**, L Velasco, and M. Ruiz, "Big Data Analytics for the Virtual Network Topology Reconfiguration Use Case," in Proceedings of the International Conference on Transparent Optical Networks (ICTON), 2016.

[Mo16] **F. Morales**, M. Ruiz, and L. Velasco, "Virtual Network Topology Reconfiguration based on Big Data Analytics for Traffic Prediction," in Proceedings of the Optical Fiber Communication Conference (OFC), 2016.

10.2.3 Book Chapters

[Mo17-7] **F. Morales**, M. Ruiz and L. Velasco, "Chapter 11 - Virtual network topology design and reconfiguration", in *Provisioning, Recovery and In-operation Planning in Elastic Optical Networks*, ISBN 978-1-119-33856-7, Wiley, 2017.

10.2.4 Other Works Not Included in This Thesis

[Ve17] L. Velasco, A. P. Vela, **F. Morales**, and M. Ruiz, "Designing, Operating and Re-Optimizing Elastic Optical Networks," (Invited Tutorial) in Journal of Lightwave Technology, vol. 35, pp. 513-526, 2017.

10.3 List of Research Projects

10.3.1 European Funded Projects

- **METRO-HAUL:** "METRO High bandwidth, 5G application-aware optical network, with edge storage, compUte and low Latency", grant agreement no. 761727, 2017-2020.

10.3.2 Spanish Funded Projects

- **SYNERGY**: "Service-oriented hYbrid optical NEtwork and cloud infrastructuRe featuring high throuGhput and ultra-low latencY", Ref: TEC2014-59995-R, 2015-2017.

10.3.3 Pre-doctoral Scholarship

- Pre-doctoral scholarship "FPI-UPC 275-701" funded by the Computer Architecture Department, UPC, 2016-2017.
- Pre-doctoral scholarship "FI-DGR 2017" funded by "Agència de Gestió d'Ajuts Universitaris i de Recerca" (AGAUR), Generalitat de Catalunya, 2017-2018.

10.4 Topics for Further Research

In this thesis we present the first step towards cognitive in-operation planning, focusing on VNT reconfiguration based on traffic prediction and on procedures to estimate quality traffic flow predictive models. In light of the results obtained, we encourage further research on other cognitive in-operation planning applications both in the optical and in the MPLS layer.

In particular, for the subject of VNT reconfiguration, decision-making techniques need to be explored to both automatically decide when to reconfigure the VNT and to automatically tune the reconfiguration parameters based on learning from past experience. From the data analytics point of view, the detection of obsolete predictive models and the correct discrimination between persistent (concept-drift) and temporary (anomalies) traffic variations is of high importance to ensure the correctness of the predictive traffic models and shall be explored as well.

List of Acronyms

ABNO	Application-Based Network Operations
ACF	Auto-Correlation Function
AIC	Akaike Information Criterion
ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	Auto-Regressive Integrated Moving Average
AS	Autonomous System
BER	Bit Error Rate
BQPSK	Binary Quadrature Phase-Shift Keying
BV	Bandwidth-Variable
BVT	Bandwidth-Variable Transponder
CAGR	Compounded Annual Growth Rate
CAPEX	Capital Expenditures
CL	Candidate List
CLI	Command-Line Interface
CP	Connection Point
CV	Coefficient of Variation
DB	Database
DC	Datacenter
DWDM	Dense Wavelength Division Multiplexing
EON	Elastic Optical Network
FEC	Forward Equivalence Class
FT	Fixed Transponder

Gb	Gigabit
GCF	Greedy Cost Function
GCO	Grup de Comunicacions Òptiques
GMPLS	Generalised Multi-Protocol Label Switching
GRASP	Greedy-Randomised Adaptive Search Procedure
HTTP	Hypertext Transfer Protocol
IAT	Inter-Arrival Time
IETF	Internet Engineering Task Force
ILP	Integer Linear Programming
ILS	Iterated Local Search
IP	Internet Protocol
IPFIX	IP Flow Information Export protocol
ITU-T	International Telecommunications Union (Telecom. standardization sector)
JSON	Javascript Object Notation
KB	Kilobyte
KDD	Knowledge Discovery in Data
LER	Label Edge Router
LP	Linear Programming
LP-SA	Link-Path Slot Assignment
LSP	Label-Switched Path
LSR	Label Switch Router
MAC	Media Access Control
MB	Megabyte
MILP	Mixed Integer Linear Programming
MPLS	Multi-Protocol Label Switching
NBI	Northbound Interface
NLP	Non-Linear Programming
NMS	Network Management System
NP	Non-deterministically Polynomial
OAA	Observe, Analyse, Act
OAM	Operations, Administrations and Maintenance
OD	Origin-Destination
ONF	Open Networking Foundation

OPEX	OPerational EXpenditures
OSS	Operations Support System
OVS	Open Virtual Switch
PCE	Path Computation Element
PCEP	Path Computation Element communication Protocol
PM	Provisioning Manager
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase-Shift Keying
RAM	Random Access Memory
RCL	Restricted Candidate List
REST	REpresentational State Transfer
RFC	Request For Comments
RMSA	Routing, Modulation format and Spectrum Allocation
ROADM	Reconfigurable Optical Add-Drop Multiplexer
RSA	Routing and Spectrum Allocation
RSS	Residual Sum of Squares
RWA	Routing and Wavelength Assignment
SBI	Southbound Interface
SBVT	Sliceable Bandwidth Variable Transponder
SDN	Software Defined Network
SNC	Subnetwork Connection
SNMP	Simple Network Management Protocol
SP	Shortest Path
SRP	Stateful Request Parameters
TCO	Total Cost of Ownership
TCP	Termination Connection Point
TE	Traffic Engineering
TED	Traffic Engineering Database
TTT	Time To Target
VM	Virtual Machine
VNT	Virtual Network Topology
VNTM	Virtual Network Topology Manager
WDM	Wavelength Division Multiplexing

WSS Wavelength Selective Switch

References

[5G] 5GPPP Working Group on Network Management and QoS, "Cognitive Network Management for 5G", 2016.

[Ag15] A. Aguado, M. Davis, S. Peng, M. V. Alvarez, V. López, T. Szyrkowiec and R. Casellas, "Dynamic Virtual Network Reconfiguration over SDN Orchestrated Multi-Technology Optical Transport Domains," in Proceedings of the European Conference on Optical Communications (ECOC), 2015.

[Agr09] F. Agraz, L. Velasco, J. Perelló, M. Ruiz, S. Spadaro, G. Junyent, and J. Comellas, "Design and Implementation of a GMPLS-Controlled Grooming-capable Optical Transport Network," in Journal of Optical Communications and Networking, vol. 1, pp. A258-A269, 2009.

[Ai07] R. Aiex, M.G.C. Resende and C.C. Ribeiro, "TTT plots: a Perl program to create time-to-target plots," in Springer Optimization Letters, vol. 1, pp. 355-366, 2007.

[Al16] R. Alvizu, X. Zhao, G. Maier, Y. Xu and A. Pattavina, "Energy aware optimization of mobile metro-core network under predictable aggregated traffic patterns," in Proceedings of the International Conference on Communications (ICC), 2016.

[Ap10] R. Aparicio-Pardo, P. Pavon-Marino, N. Skorin-Kapov, B. Garcia-Manrubia, and J. Garcia-Haro, "Algorithms for Virtual Topology Reconfiguration under Multi-Hour Traffic using Lagrangian Relaxation and Tabu Search approaches," in Proceedings of the International Conference on Transparent Optical Networks (ICTON), 2010.

[Ap12] R. Aparicio-Pardo, N. Skorin-Kapov, P. Pavon-Marino, and B. Garcia-Manrubia, "(Non-)Reconfigurable Virtual Topology Design Under Multihour Traffic in Optical Networks," in Transactions on Networking, vol. 20, no. 5, pp. 1567-1580, 2012.

[As14] M. T. Asif, J. Dauwels, C. Y. Goh, A. Oran, E. Fathi, M. Xu and P. Jaillet, "Spatiotemporal patterns in large-scale traffic speed prediction," in Transactions on Intelligent Transportation Systems, vol. 15(2), pp. 794-804, 2014.

[AV16] A. P. Vela, A. Via, F. Morales, M. Ruiz, and L. Velasco, "Traffic generation for telecom cloud -based simulation," in Proceedings of the International Conference on Transparent Optical Networks (ICTON), 2016.

[AV17-1] A. P. Vela, M. Ruiz, L. Velasco, "Distributing Data Analytics for Efficient Multiple Traffic Anomalies Detection," in Elsevier Computer Communications, vol. 107, pp. 1-12, 2017.

[AV17-2] A. P. Vela, M. Ruiz, F. Fresi, N. Sambo, F. Cugini, L Velasco, and P. Castoldi, "Early Pre-FEC BER Degradation Detection to Meet Committed QoS," in Proceedings of the Optical Fiber Communication Conference (OFC), 2017.

[Ba04] C. Barakat, P. Thiran, G. Iannaccone, C. Diot and P. Owezarski, "Modeling Internet backbone traffic at the flow level," in Transactions on Signal processing, vol. 51(8), pp. 2111-2124, 2003.

[Be92] D. P. Bertsekas, R. G. Gallager and P. Humblet, *Data networks*, 2nd edition, Prentice-Hall, 1992.

[Ca16] A. Castro, L. Velasco, Ll. Gifre, C. Chen, J. Yin, Z. Zhu, R. Proietti and S. J. B. Yoo, "Brokered orchestration for end-to-end service provisioning across heterogeneous multi-operator (Multi-AS) optical networks," in Journal of Lightwave Technology, vol. 34(23), pp. 5391-5400, 2016.

[CaPhd] A. Castro, *Off-Line and In-Operation Optical Core Networks Planning*, Doctoral dissertation, Computer Architecture Department, Universitat Politècnica de Catalunya, 2014.

[Cas] Apache Cassandra: http://cassandra.apache.org

[Ch06] T. M. Chen, "Network Traffic Modeling," in *Handbook of Computer Networks vol. 3*, Wiley, 2007.

[Cisco17] "Cisco Visual Networking Index," Cisco, 2017.

[Cisco99] "MPLS Traffic Engineering", Cisco white paper, 1999.

[El01] A. Elwalid, C. Jin, S. Low and I. Widjaja, "MATE: MPLS adaptive traffic engineering," in Proceedings of the International Conference on Computer Communications (INFOCOM), 2001.

[Em08] F. Emmert-Streib and M. Dehmer, *Information Theory and Statistical Learning*, Springer Science and Business Media, 2008.

[Fe02] P. Festa and M.G. Resende, "GRASP: An annotated bibliography," in Essays and Surveys on Metaheuristics, vol. 6, pp. 325-367, 2002.

[Fe15] N. Fernández, R. Durán, D. Siracusa, A. Francescon, I. de Miguel, E. Salvadori, J. Aguado and R. Lorenzo, "Virtual Topology Reconfiguration in Optical Networks by Means of Cognition: Evaluation and Experimental Validation," in Journal of Optical Communications and Networking, vol. 7, pp. A162 - A173, 2015.

[Ga14] J. Gama, I. Zliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," in Journal of Computing Surveys, vol. 46, p. 44, 2014.

[Gen03] A. Gençata and B. Mukherjee, "Virtual-Topology Adaptation for WDM Mesh Networks Under Dynamic Traffic," in Transactions on Networking, vol. 11, pp. 236-247, 2003.

[Gi14] Ll. Gifre, F. Paolucci, A. Aguado, R. Casellas, A. Castro, F. Cugini, P. Castoldi, L. Velasco, and V. López, "Experimental Assessment of In-Operation Spectrum Defragmentation," in Springer Photonic Network Communications, vol. 27, pp. 128-140, 2014.

[Gi15] Ll. Gifre, F. Paolucci, L. Velasco, A. Aguado, F. Cugini, P. Castoldi, and V. López, "First Experimental Assessment of ABNO-driven In-Operation Flexgrid Network Re-Optimization," in Journal of Lightwave Technology, vol. 33, pp. 618-624, 2015.

[Gi16-1] Ll. Gifre, L. M. Contreras, V. Lopez, and L. Velasco, "Big Data Analytics in Support of Virtual Network Topology Adaptability," in Proceedings of the Optical Fiber Communication Conference (OFC), 2016.

[Gi17] Ll. Gifre, A. P. Vela, M. Ruiz, J. López de Vergara, and L. Velasco, "Experimental Assessment of Node and Control Architectures to Support the Observe-Analyze-Act Loop," in Proceedings of the Optical Fiber Communication Conference (OFC), 2017.

[GiPhD] Ll. Gifre, *In-Operation Planning in Flexgrid Optical Core Networks*, Doctoral dissertation, Computer Architecture Department, Universitat Politècnica de Catalunya, 2016.

[GiPhD] Ll. Gifre, *In-Operation Planning in Flexgrid Optical Networks*, Doctoral dissertation, Computer Architecture Department, Universitat Politècnica de Catalunya, 2016.

[Gr06] E. W. Grafarend, *Linear and Nonlinear Models: Fixed Effects, Random Effects, and Mixed Models*, Walter de Gruyter, 2006.

[Gro08] D. Gross, J. F. Shortle, J. M. Thompson and C. M. Harris, *Fundamentals of Queuing Theory*, 4th edition, Wiley, 2008.

[Ho13] R. V. Hoog, *Introduction to Mathematical Statistics*, Pearson, 2013.

[Hol03] M. Hollick, T. Krop, J. Schmitt, H. P. Huth and R. Steinmetz, "A hybrid workload model for wireless metropolitan area networks," in Proceedings og the Vehicular Technology Conference, 2003.

[It95] H. Ito, "Computational complexity of multicommodity flow problems with uniform assignment of flow," in Electronics and Communications in Japan vol. 78(8), pp. 52-62, 1995.

[Ko13] S. K. Korotky, "Semi-Empirical Description and Projections of Internet Traffic Trends Using a Hyperbolic Compound Annual Growth Rate," in Bell Labs Technical Journal, vol. 18, no 3, pp. 5-21, 2013.

[Lau00] F. Lau, S. H. Rubin, M. H. Smith and L. Trajkovic, "Distributed denial of service attacks," in Proceedings of the International Conference on Systems, Man, and Cybernetics, 2000.

[Lee14] D. Lee, S. Zhou, X. Zhong, Z. Niu, X. Zhou and H. Zhang, "Spatial modeling of the traffic density in cellular networks," in Wireless Communications, vol. 21(1), pp. 80-88, 2014.

[Lo03] H.R. Lourenço, O.C. Martin and T. Stutzle, "Iterated local search," in International Series in Operations Research and Management Science, pp. 321-54, 2003.

[Lu11] H Lütkepohl, "Forecasting aggregated time series variables: A survey," in Journal of Business Cycle Measurement and Analysis, vol. 2010, pp. 1-26, 2011.

[Ma08] E. C. Malthouse, K. M. Derenthal, "Improving predictive scoring models through model aggregation," in Elsevier Journal of Interactive Marketing, vol. 22, pp. 51-68, 2008.

[Mal14] M. Malboubi, L. Wang, C. N. Chuah and P. Sharma, "Intelligent SDN based traffic (de)aggregation and measurement paradigm (iSTAMP)," in Proceedings of the Internation Conference on Computer Communications (INFOCOM), 2014.

[Me10] M. Medhat Gaber, A. Zaslavsky, S. Krishnaswamy, "Data Stream Mining," in Data Mining and Knowledge Discovery Handbook, Springer, pp. 759-787, 2010.

[Oh10] Y. Ohsita, T. Miyamura, S. Arakawa, S. Ata, E. Oki, K. Shiomoto and M. Murata, "Gradually Reconfiguring VNT Based on Estimated Traffic Matrices," in Transactions on Networking, vol. 18, pp. 177-189, 2010.

[OMNeT] OMNeT++: http://www.omnetpp.org.

[ONF] Open Networking Foundation [Online]. Available: https://www.opennetworking.org/.

[OpenFlow] OpenFlow [Online]. Available: http://www.openflow.org/.

[Pa10] P. Pavon-Marino, R. Aparicio-Pardo, B. Garcia-Manrubia, and N. Skorin-Kapov, "Virtual topology design and flow routing in optical networks under multihour traffic demand," in Photonic Network Communications, vol. 19, no. 1, pp. 42-54, Feb. 2010.

[Pe13] O. Pedrola, M. Ruiz, L. Velasco, D. Careglio, O. González de Dios, and J. Comellas, "A GRASP with path-relinking heuristic for the survivable IP/MPLS-over-WSON multi-layer network optimization problem," in Elsevier Computers and Operations Research, vol. 40, pp. 3174-3187, 2013.

[Pro18] R. Proietti, X. Chen, A. Castro, G. Liu, H. Lu, K. Zhang, J. Guo, Z. Zhu, L. Velasco, and S. J. Ben Yoo, "Experimental Demonstration of Cognitive Provisioning and Alien Wavelength Monitoring in Multi-domain EON," accepted in Proceedings of the Optical Fiber Communication Conference (OFC), 2018.

[Ra09] R. Randhawa, C. A. Shaffer, J. J. Tyson, "Model aggregation: a building-block approach to creating large macromolecular regulatory networks," in Oxford Academic Bioinformatics, vol. 25, no 24, p. 3289-3295, 2009.

[RFC1157] J. Case, M. Fedor, M. Schoffstall and J. Davin, "A Simple Network Management Protocol (SNMP)," IETF RFC1157, 1990.

[RFC3031] E. Rosen, A. Viswanathan, A. Viswanathan, "Multiprotocol Label Switching Architecture," IETF RFC3031, 2001.

[RFC3535] J. Schoenwaelder, "Overview of the 2002 IAB Network Management Workshop," IETF RFC3535, 2002.

[RFC3945] E. Mannie, "Generalized Multi-Protocol Label Switching (GMPLS) Architecture," IETF RFC3945, 2004.

[RFC4655] A. Farrel, J.-P. Vasseur, and J. Ash, "A Path Computation Element (PCE)-Based Architecture," IETF RFC4655, 2006.

[RFC5212] K. Shiomoto, D. Papadimitriou, JL. Le Roux, M. Vigoureux, and D. Brungard, "Requirements for GMPLS-Based Multi-Region and Multi-Layer Networks (MRN/MLN)," IETF RFC5212, 2008.

[RFC5623] E. Oki, T. Takeda, JL. Le Roux, and A. Farrel, "Framework for PCE-Based Inter-Layer MPLS and GMPLS Traffic Engineering," IETF RFC5623, 2009.

[RFC7011] B. Claise, B. Trammell, P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol," IETF RFC 7011, 2013.

[RFC7491] D. King and A. Farrel, "A PCE-Based Architecture for Application-Based Network Operations," IETF RFC7491, 2015.

[RFC8261] E. Crabbe, I. Minei, J. Medved and R. Varga, "PCEP Extensions for Stateful PCE," IETF RFC 8261, 2017.

[Ri13] C. C.Ribeiro, I. Rosseti and R. C. Souza, "Probabilistic stopping rules for GRASP heuristics and extensions," in International Transactions in Operational Research, vol. 20(3), pp. 301-323, 2013.

[Ro08] J. P. Roorda and B. Collings, "Evolution to colorless and directionless ROADM architectures," in Optical Fiber Conference/National Fiber Optic Engineers Conference, 2008.

[Ru14] M. Ruiz, L. Velasco, A. Lord, D. Fonseca, M. Pióro, R. Wessäly, and J.P. Fernández-Palacios, "Planning Fixed to Flexgrid Gradual Migration: Drivers and Open Issues," in Communications Magazine, vol. 52, pp. 70-76, 2014.

[Ru16] M. Ruiz, M. Germán, L. M. Contreras, and L. Velasco, "Big Data-backed Video Distribution in the Telecom Cloud," in Elsevier Computer Communications, vol. 84, pp. 1-11, 2016.

[RyuSDN] Ryu SDN framework: https://osrg.github.io/ryu

[Sa16] N. Sambo, F. Cugini, A. Sgambelluri, and P. Castoldi, "Monitoring Plane Architecture and OAM Handler," in Journal of Lightwave Technology, vol. 34, pp. 1939-1945, 2016.

[SDN] C. S. Li and W. Liao, "Software defined networks," in Communications Magazine, vol. 51(2), pp. 113-113, 2013.

[Sga16] A. Sgambelluri, F. Paolucci, A. Giorgetti, F. Cugini and P. Castoldi, "Experimental Demonstration of Segment Routing," in Journal of Lightwave Technology, vol. 34, pp. 205-212, 2016.

[Spark] Apache Spark: http://spark.apache.org

[St82] K. Steiglitz and C. H. Papadimitriou, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, New Jersey, 1982.

[Ta07] V. Tabatabaee, A. Kashyap, B. Bhattacharjee, R. J. La, and M. A. Shayman, "Robust Routing with Unknown Traffic Matrices," in Proceedings of the International Conference on Computer Communications (INFOCOM), 2007.

[Tr13] B. Trammell and B. Claise, "Applying IPFIX to network Measurement and Management," IETF tutorial. Available on https://www.ietf.org/slides/slides-edu-ipfix-00.pdf

[Ve12] L. Velasco, M. Klinkowski, M. Ruiz, and J. Comellas, "Modeling the Routing and Spectrum Allocation Problem for Flexgrid Optical Networks," in Springer Photonic Network Communications, vol. 24, pp. 177-186, 2012.

[Ve13] L. Velasco, P. Wright, A. Lord, and G. Junyent, "Saving CAPEX by Extending Flexgrid-based Core Optical Networks toward the Edges," (Invited Paper) in Journal of Optical Communications and Networking, vol. 5, pp. A171-A183, 2013.

[Ve14.1] L. Velasco, D. King, O. Gerstel, R. Casellas, A. Castro, and V. López, "In-Operation Network Planning," in IEEE Communications Magazine, vol. 52, pp. 52-60, 2014.

[Ve14.2] L. Velasco, A. Castro, M. Ruiz, and G. Junyent, "Solving Routing and Spectrum Allocation Related Optimization Problems: from Off-line to In-Operation Flexgrid Network Planning," in Journal of Lightwave Technology, vol. 32(16), pp. 2780-2795, 2014.

[Ve14.3] L. Velasco, A. Asensio, J. Ll. Berral, E. Bonetto, F. Musumeci, V. López, "Elastic Operations in Federated Datacenters for Performance and Cost Optimization," in Computer Communications, vol. 50, pp. 142-151, 2014.

[Ve15] L. Velasco and Ll. Gifre, "iONE: A Workflow-Oriented ABNO Implementation," in Proceedings of the International Photonics in Switching Conference, 2015.

[Ve16] L. Velasco, A. P. Vela, F. Morales, and M. Ruiz, "Designing, Operating and Re-Optimizing Elastic Optical Networks," in Journal of Lightwave Technology, vol. 35, pp. 513-526, 2017.

[Wei94] W. Wei, *Time series analysis*, Addison-Wesley publications, 1994.

[Wi98] W. Willinger and V. Paxson, "Where mathematics meets the Internet," in Notices of the AMS, vol. 45, pp. 961-970, 1998.

[Wu15] X. Wu, Z. Lü, Q. Guo & T. Ye, "Two-level iterated local search for WDM network design problem with traffic grooming," in Elsevier Applied Soft Computing, vol. 37, pp. 715-724, 2015.

[Yu09] X. Yuan, N. Chen, D, Wang, G. Xie, and D. Zhang, "Traffic prediction models of traffics at application layer in metro area network," in Journal of Computer Research and Development, vol. 3, p. 13, 2009.

[Zh03] Y. Zhang, M., Roughan, N., Duffield and A. Greenberg, "Fast accurate computation of large-scale IP traffic matrices from link loads," in SIGMETRICS Performance Evaluation Review, vol. 31, no. 1, pp. 206-217, 2003.

[Zha13] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang and Y. Guan, "Network traffic classification using correlation information", in Transactions on Parallel and Distributed Systems, vol. 24, pp. 104-117, 2013.

[Zhe17] L. Zheng, D. Liu, W. Liu, Z. Liu, Z. Li, T. Wu, "A Data Streaming Algorithm for Detection of Superpoints with Small Memory Consumption," in Communications Letters, vol. PP, pp.1-1, 2017.

[Zu03] M. Zukerman, T. D. Neame and R. G. Addie, "Internet traffic modeling and future technology implications," in Proceedings of the Computer and Communications Conference, 2003.