UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

# *Distributed collaborative knowledge management for optical networks*

**Fatemehsadat Tabatabaeimehr**

Universitat Politècnica de Catalunya

Optical Communications Group

# Distributed Collaborative Knowledge Management for Optical Networks

**Fatemehsadat Tabatabaeimehr**

Advisor:

Dr. Luis Velasco

Co-advisor:

Dr. Jaume Comellas

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

**Philosophy Doctor**

April 19, 2022

Optical Communications Group (GCO)

Universitat Politècnica de Catalunya (UPC)

C/ Jordi Girona, 1-3

Campus Nord, D4-213

08034 Barcelona, Spain

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola de Doctorat

# Acta de calificación de tesis doctoral

**Curso académico:**

Nombre y apellidos

Programa de doctorado

Unidad estructural responsable del programa

## Resolución del Tribunal

Reunido el Tribunal designado a tal efecto, el doctorando / la doctoranda expone el tema de la su tesis doctoral titulada _____

_____.

Acabada la lectura y después de dar respuesta a las cuestiones formuladas por los miembros titulares del tribunal, éste otorga la calificación:

☐ NO APTO ☐ APROBADO ☐ NOTABLE ☐ SOBRESALIENTE

| | |
|---|---|
| (Nombre, apellidos y firma)<br>Presidente/a | (Nombre, apellidos y firma)<br>Secretario/a |

| | | |
|---|---|---|
| (Nombre, apellidos y firma)<br>Vocal | (Nombre, apellidos y firma)<br>Vocal | (Nombre, apellidos y firma)<br>Vocal |

_____, _____ de _____ de _____

El resultado del escrutinio de los votos emitidos por los miembros titulares del tribunal, efectuado por la Escuela de Doctorado, a instancia de la Comisión de Doctorado de la UPC, otorga la MENCIÓN CUM LAUDE:

☐ SÍ ☐ NO

| | |
|---|---|
| (Nombre, apellidos y firma)<br>Presidente de la Comisión Permanente de la Escuela de Doctorado | (Nombre, apellidos y firma)<br>Secretaria de la Comisión Permanente de la Escuela de Doctorado |

Barcelona a _____ de _____ de _____

# Acknowledgements

This dissertation is the milestone of my work over last years both academically and professionally. It was an invaluable experience for me and it would not have been possible to do without the guidance that I received from many people.

Foremost, I would like to express my gratitude to my advisors, Prof. Luis Velasco and Prof. Jaume Comellas that gave me the opportunity to join GCO group and develop my research and knowledge among professional team members. I appreciate your wise advice and patience. To Luis, for pushing me to be the best and teaching me the importance of doing high-quality research. I would like also to express my gratitude to Dr. Marc Ruiz, for his unwavering help and guidance during the countless hours devoted to the teaching and research behind this thesis.

Moreover, I am very grateful to Dr. Ramon Aparicio and Dr. Alberto Castro who have reviewed this PhD thesis. I wish to thank all the former and current colleagues and friends especially Dr. Behnam Shariati, Dr. Alba Pérez, and Sima Barzegar which I was lucky to share many moments during my program.

Finally, I would like to acknowledge the Agència de Gestió d'Ajuts Universitaris i de Recerca (AGAUR) for the financial supports.

This thesis is dedicated to my dearests. To my parents who I am eternally gratitude for their unconditional love and tremendous support. To my sisters and brother, Faezeh, Atyeh and Mohammad Javad for the encouragement and motivation always give me.

# Abstract

Network automation has been long time envisioned. In fact, the Telecommunications Management Network (TMN), defined by the International Telecommunication Union (ITU), is a hierarchy of management layers (network element, network, service, and business management), where high-level operational goals propagate from upper to lower layers.

The network management architecture has evolved with the development of the Software Defined Networking (SDN) concept that brings programmability to simplify configuration (it breaks down high-level service abstraction into lower-level device abstractions), orchestrates operation, and automatically reacts to changes or events. Besides, the development and deployment of solutions based on Artificial Intelligence (AI) and Machine Learning (ML) for making decisions (control loop) based on the collected monitoring data enables network automation, which targets at reducing operational costs.

AI/ML approaches usually require large datasets for training purposes, which are difficult to obtain. The lack of data can be compensated with a collective self-learning approach. In this thesis, we go beyond the aforementioned traditional control loop to achieve an efficient knowledge management (KM) process that enhances network intelligence while bringing down complexity.

In this PhD thesis, we propose a general architecture to support KM process based on four main pillars, which enable creating, sharing, assimilating and using knowledge. Next, two alternative strategies based on model inaccuracies and combining model are proposed. To highlight the capacity of KM to adapt to different applications, two use cases are considered to implement KM in a purely centralized and distributed optical network architecture. Along with them, various policies are considered for evaluating KM in data- and model- based strategies. The results target to minimize the amount of data that need to be shared and reduce the convergence error.

We apply KM to multilayer networks and propose the PILOT methodology for modeling connectivity services in a sandbox domain. PILOT uses active probes

deployed in Central Offices (COs) to obtain real measurements that are used to tune a simulation scenario reproducing the real deployment with high accuracy. A simulator is eventually used to generate large amounts of realistic synthetic data for ML training and validation.

We apply KM process also to a more complex network system that consists of several domains, where intra-domain controllers assist a broker plane in estimating accurate inter-domain delay. In addition, the broker identifies and corrects intra-domain model inaccuracies, as well as it computes an accurate compound model. Such models can be used for quality of service (QoS) and accurate end-to-end delay estimations.

Finally, we investigate the application on KM in the context of Intent-based Networking (IBN). Knowledge in terms of traffic model and/or traffic perturbation is transferred among agents in a hierarchical architecture. This architecture can support autonomous network operation, like capacity management.

It shall be mentioned that part of the work reported in this thesis has been done within the framework of European and National projects. Specifically, the H2020 METRO-HAUL and B5G-OPEN funded by the European Commission, and the MINECO TWINS and AEI IBON, both funded by the Spanish Ministry of Economy, Industry and Competitiveness.

# Resumen

La automatización de la red se ha concebido desde hace mucho tiempo. De hecho, la red de gestión de telecomunicaciones (TMN), definida por la Unión Internacional de Telecomunicaciones (ITU), es una jerarquía de capas de gestión (elemento de red, red, servicio y gestión de negocio), donde los objetivos operativos de alto nivel se propagan desde las capas superiores a las inferiores.

La arquitectura de gestión de red ha evolucionado con el desarrollo del concepto de redes definidas por software (SDN) que brinda capacidad de programación para simplificar la configuración (descompone la abstracción de servicios de alto nivel en abstracciones de dispositivos de nivel inferior), organiza la operación y reacciona automáticamente a los cambios o eventos. Además, el desarrollo y despliegue de soluciones basadas en inteligencia artificial (IA) y aprendizaje automático (ML) para la toma de decisiones (bucle de control) en base a los datos de monitorización recopilados permite la automatización de la red, que tiene como objetivo reducir costes operativos.

AI/ML generalmente requieren un gran conjunto de datos para entrenamiento, los cuales son difíciles de obtener. La falta de datos se puede compensar con un enfoque de autoaprendizaje colectivo. En esta tesis, vamos más allá del bucle de control tradicional antes mencionado para lograr un proceso eficiente de gestión del conocimiento (KM) que mejora la inteligencia de la red al tiempo que reduce la complejidad.

En esta tesis doctoral, proponemos una arquitectura general para apoyar el proceso de KM basada en cuatro pilares principales que permiten crear, compartir, asimilar y utilizar el conocimiento. A continuación, se proponen dos estrategias alternativas basadas en inexactitudes del modelo y modelo de combinación. Para resaltar la capacidad de KM para adaptarse a diferentes aplicaciones, se consideran dos casos de uso para implementar KM en una arquitectura de red óptica puramente centralizada y distribuida. Junto a ellos, se consideran diversas políticas para evaluar KM en estrategias basadas en datos y modelos. Los resultados apuntan a

minimizar la cantidad de datos que deben compartirse y reducir el error de convergencia.

Aplicamos KM a redes multicapa y proponemos la metodología PILOT para modelar servicios de conectividad en un entorno aislado. PILOT utiliza sondas activas desplegadas en centrales de telecomunicación (CO) para obtener medidas reales que se utilizan para ajustar un escenario de simulación que reproducen un despliegue real con alta precisión. Un simulador se utiliza finalmente para generar grandes cantidades de datos sintéticos realistas para el entrenamiento y la validación de ML.

Aplicamos el proceso de KM también a un sistema de red más complejo que consta de varios dominios, donde los controladores intra-dominio ayudan a un plano de bróker a estimar el retardo entre dominios de forma precisa. Además, el bróker identifica y corrige las inexactitudes de los modelos intra-dominio, así como también calcula un modelo compuesto preciso. Estos modelos se pueden utilizar para estimar la calidad de servicio (QoS) y el retardo extremo a extremo de forma precisa.

Finalmente, investigamos la aplicación en KM en el contexto de red basada en intención (IBN). El conocimiento en términos de modelo de tráfico y/o perturbación del tráfico se transfiere entre agentes en una arquitectura jerárquica. Esta arquitectura puede soportar el funcionamiento autónomo de la red, como la gestión de la capacidad.

Cabe mencionar que parte del trabajo reportado en esta tesis se ha realizado en el marco de proyectos europeos y nacionales. En concreto, H2020 METRO-HAUL y B5G-OPEN financiados por la Comisión Europea, y MINECO TWINS y AEI IBON, ambos financiados por el Ministerio de Economía, Industria y Competitividad de España.

# Table of Contents

Page

# List of Figures

Page

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

The optical network is being extended toward the edges of operators' networks [Ve13], fostered not only by the increased amount of traffic coming from current and future access segment, but also by the stringent Key Performance Indicators (KPI) that they need to support, like *low latency* and *high reliability*. In fact, more and more connectivity services are requiring not only stringent, but also more *predictable* Quality of Service (QoS) performance, such as throughput and latency.

Accelerated by such requirements, new solutions for the control and orchestration of the optical transport network are being proposed (see, e.g., [Fi19]). Such services are supported by a packet layer on top of an optical network, where the latter covers core and metro segments and provides high capacity with low latency and high reliability.

The added complexity, in addition to highly dynamic traffic, requires the network operation to be automated. In this regard, autonomous control loops based on Machine Learning (ML) techniques [Ve18.1] have been proposed aiming at reducing human intervention as a way to minimize network operational costs. In general, an autonomous control loop uses *knowledge discovered* during a ML training phase to predict (near) future network conditions, so as to proactively prepare resources to deal with them (*decision-making*).

Considering that *knowledge usage* and decision making are needed not only at the controller level, but also at the local node/subsystem level, the control plane should be designed to support such variety of use cases and scenarios of autonomous networking.

However, enough real data to produce accurate ML models is rarely available owing to a plethora of reasons, like the existing legal and regulatory context that limits the availability of real network performance measurement, as well as the difficulty to obtain training datasets belonging to specific pre-commercial and commercial technologies and use them in current and forecasted scenarios.

In view of that, the authors in [Ve19.1] proposed a learning life-cycle to facilitate ML deployment in real operator networks. In particular, they added a ML training phase to be carried out after detecting model inaccuracies (e.g., in the form of prediction errors), being this the basis of self-learning to progressively improve the ML models deployed in the network. Such improvement can be made faster in the case of the model is being used by several agents, which can share model's inaccuracies among them; they called this as collective self-learning. It was demonstrated that collective self-learning outperforms individual strategies. However, because the size of the training dataset might be large to reach high-accuracy and robustness, (data-based) collective self-learning increases data to be stored and to be exchanged among agents.

## 1.2 Goals of the thesis

In light of the above, this Ph.D. thesis goes further and targets at completing the *knowledge management* (KM) process for truly autonomous network operation. The KM process entails creating and sharing knowledge and it has been applied to achieve organizational objectives, like continuous improvement of an organization. Those *learning organizations* are able to adapt quickly and effectively to be superior to the competitors in their field or market [Se90].

This Ph.D. thesis focuses on studying the application of KM to operator multi-layer networks in single domain or multi-domain multi-operator scenarios. Four specific goals are defined to achieve this main goal:

**G.1 - Knowledge management (KM) in single domain networks**

This goal targets at providing structure to collective self-learning aiming at improving models error convergence time, as well as at minimizing the amount of data being shared and stored.

In order to fully achieve this goal, we need to tackle two specific sub-goals:

**G1.1 - KM process definition and architecture**: In this sub-goal, we design KM defining the main processes to allow autonomous learning. In addition, a KM architecture will be proposed to support a wide variety of use cases.

**G1.2 - Knowledge sharing and assimilation**: specific methods to distribute and manage ML models, as well as to combining models containing new knowledge are designed, implemented, and tested.

**G.2 - KM for multi-layer networks**

Although each layer can apply KM independently, correlation between layers cannot be ignored. This goal thus, aims at adding interaction between KM agents at different network layers.

**G.3 - KM for broker-based multi-domain networks**

This goal first concentrates on the packet layer and proposes the coordination of ML capabilities between domain network controllers and the broker; abstracted data need to be exchanged in order to achieve robust and accurate models for end-to-end QoS (delay) estimation.

**G.4 - KM in Intent-based networking scenarios**

Intent-Based Networking (IBN) allows network operators to define what are their desired outcomes without specifying how they would be achieved, so it looks a natural environment for the application of ML, in particular KM. In this goal, we target at providing solutions for the cooperation among intents.

A summary of the goals of the thesis is presented Table 1-1.

*Table 1-1. Thesis goals*

<table>
<tr><th colspan="2">Goals</th></tr>
<tr><td rowspan="2">G1 -<br>KM in single domain networks</td><td>G1.1 -<br>KM process definition and architecture</td></tr>
<tr><td>G.1.2 -<br>Knowledge sharing and assimilation</td></tr>
<tr><td colspan="2">G2 -<br>KM for multi-layer networks</td></tr>
<tr><td colspan="2">G3 -<br>KM for broker-based multi-domain networks</td></tr>
<tr><td colspan="2">G4 -<br>KM in Intent-based networking</td></tr>
</table>

## 1.3 Methodology

This Ph.D. thesis assumes the architecture in Fig. 1-1a, where the data plane consists of: i) an optical layer consisting of a number of ROADMs interconnected through optical fiber links. Optical connections (*lightpaths*) can be created between two nodes in the network, ii) an overlaying packet network, where packet nodes (i.e.,

routers) are interconnected by means of logical (virtual) connections, each supported by lightpaths. Each location, named central office (CO), consists of an optical collocated with a packet node, as well as other infrastructure such as computing resources.

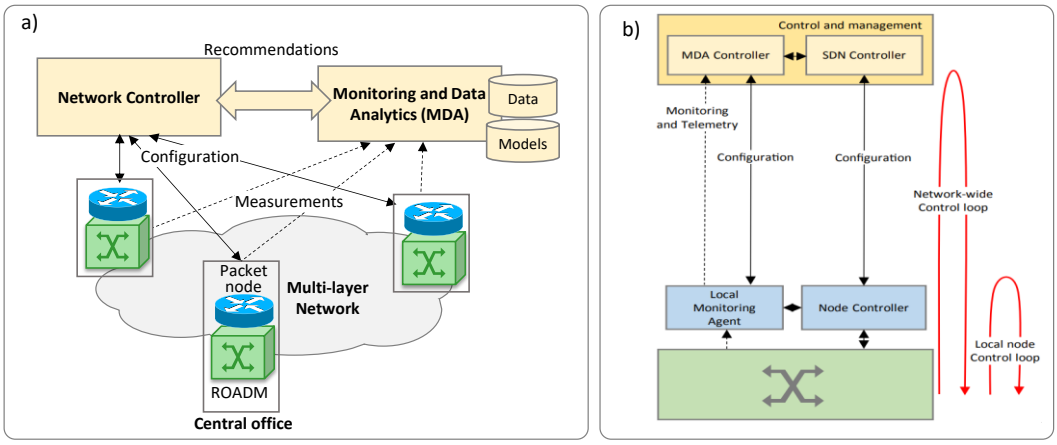


Fig. 1-1. a) General architecture and b) distributed MDA (from [Ve19.2])

The control plane includes: *i*) a Network Controller to program the network devices, and *ii*) a MDA system that collates measurements from the data plane, analyses the data and issues recommendations to the network controller. Hereafter, we assume the distributed configuration in Fig. 1-1b, where COs run local agents [Ve19.2] in order to deploy MDA processes close to where data are collected (thus allowing the implementation of local node control loops), as well as an MDA controller that collates inputs from MDA agents and allows network-wide control loops.

To carry out the studies needed to meet the goals of this thesis, the methodology illustrated in Fig. 1-2 has been followed.

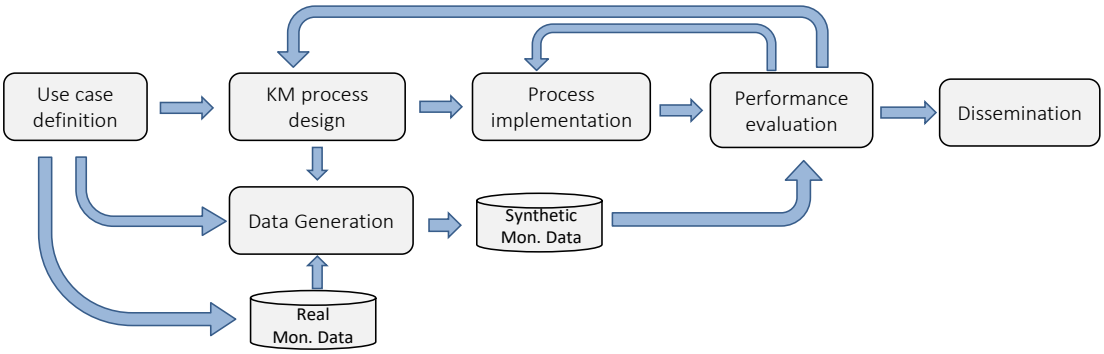


Fig. 1-2. Methodology

As a starting point, use cases are conceived. Based on the use case, the KM process is designed, and a specific data generator is developed based on real monitoring data. The algorithmic part of KM process has been implemented in R and the data synthetically generated is used for evaluation purposes and compared to a target

accuracy; improvements in both the KM process and algorithmic parts can be carried out in this phase. Finally, the results have been disseminated in international conferences and journals.

## 1.4 Thesis outline

The remainder of this research plan is organized as follows.

Chapter 2 provides the needed background on metro networks and ML and deep learning concepts.

Chapter 3 briefly reviews the state-of-the-art related to the objectives of this Ph.D. thesis, focusing and highlighting the niches to be covered.

Chapter 4 focuses on goal G.1 and introduces the KM architecture and its main features discovery, assimilation, sharing and usage. This chapter is based on the journal publication [JOCN20].

Chapter 5 relates to goal G.2 and investigates the application of KM in a multilayer network scenario. This chapter is based on one journal publication [JLT20].

Chapter 6 aims the achievement of goal G.3, where we apply KM in multi-domain network scenarios based on a broker plane. This chapter is based on the journal paper [TNSM21].

Chapter 7 concentrates on goal G.4 and is devoted to the application of KM in Intent-based networking scenarios. This chapter is based on the journal publication [JOCN22].

Finally, Chapter 8 concludes this Ph.D. thesis.

## 1.5 Contributions and Reference from the Literature

For the sake of clarity and readability, references contributing to this Ph.D. thesis are labelled using the following criteria: [<conference/journal> <Year(yy)[.autonum]>], e.g., [OFC21] or [JOCN20]; in case of more than one contribution with the same label, a sequence number is added.

The rest of the references to papers or books, both auto references not included in this Ph.D. thesis and other references from literature are labelled with the initials of the first author's surname together with its publication year, e.g., [Ve13].

# Chapter 2

# Background

The optical network is being extended toward the edges of operators' networks [Ve13], fostered not only by the increased amount of traffic coming from current and future access segment, but also by the stringent requirements that they need to support, like low latency and high reliability. The added complexity, in addition to highly dynamic traffic, requires the network operation to be automated. In this regard, autonomous *control loops* based on ML techniques [Ra18] have been proposed aiming at reducing human intervention as a way to minimize network operational costs. In general, an autonomous control loop *uses knowledge* discovered during a ML training phase to predict (near) future network conditions, so as to proactively prepare resources to deal with them (*decision-making*).

## 2.1 Infrastructure and Control Plane

This Ph.D. thesis assumes the metro infrastructure and control plane from the H2020 METRO-HAUL project (see Chapter 8.3.1). We describe next such infrastructure and the control, orchestration, and management (COM) architecture that is assumed for the single domain studies carried out in this work. The infrastructure supports several key features, including e2e latency-awareness, as well as monitoring and data analytics capabilities, to operate a partially disaggregated edge computing enabled metro optical network.

### 2.1.1 Metro Infrastructure

The METRO-HAUL infrastructure spans nodes residing in Central Offices (CO) in different geographic locations, where every node combines networking, processing, and storage resources. Such modular nodes are composed of different components

operating at different layers and technologies, and of different vendors realizing hardware and software disaggregation. METRO-HAUL nodes implement layer 0-1 (optical domain) and layer 2 transmission and switching (frame domain) and include Edge Computing capabilities provided by a local pool of computers to instantiate Virtualized Network Functions (VNF) with configurable amounts of processing, memory, and storage. Two specializations of the generic METRO-HAUL nodes are: a) Access Metro Edge nodes (AMEN) to interface with heterogeneous access technologies (5G and optical); and b) Metro Core Edge nodes (MCEN) nodes as gateways towards the core transport network and comprise core-oriented capabilities (Fig. 2-1). The nodes are controlled by a Node Agent based on NETCONF/YANG handling the integration of such disaggregated components.

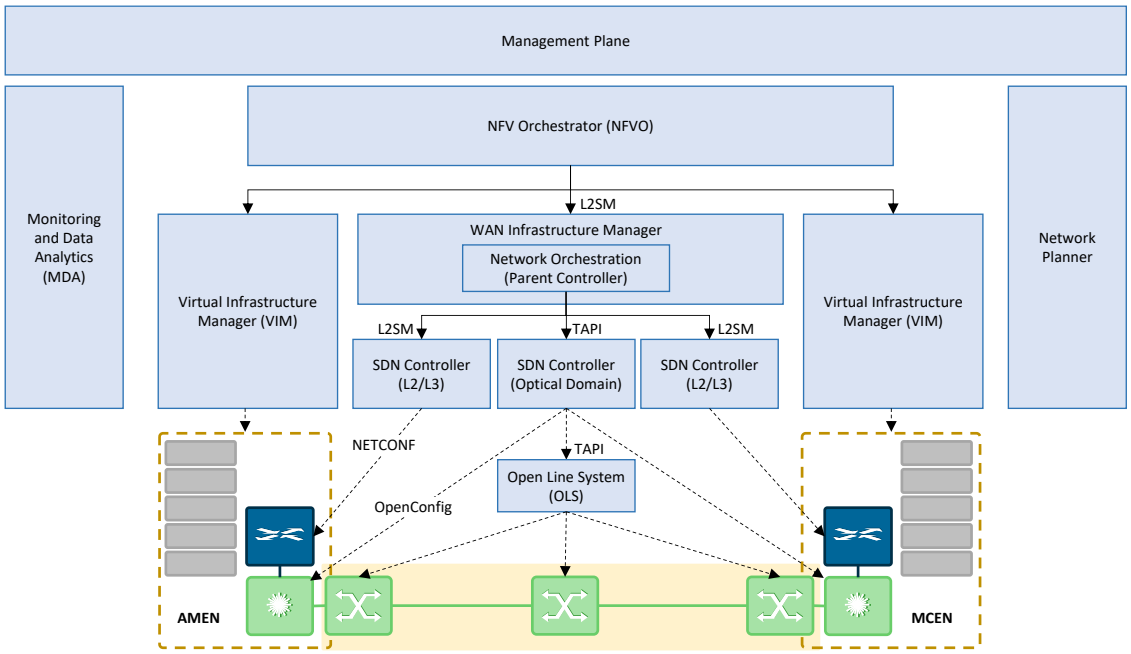


Fig. 2-1. METRO-HAUL Control, Orchestration, and Management system and network, compute, and storage infrastructure

## 2.1.2 Control, Orchestration, and Management

The previously described metro infrastructure requires a complex COM system, which includes several subsystems and interfaces among them. The COM system augments the concept of network control plane with standard interfaces operating across domains to ensure vendor inter-operability. The architecture of the METRO-HAUL COM system has evolved from its initial design and successive refinements have been made driven by feedback gathered after implementation and integration activities. The main components include the following (see Fig. 2-1).

1) The *NFV Orchestrator* (NFVO) that performs Service Orchestration (involving the functional split of the service into/amongst different VNFs and their logical

interconnection) and Resource Orchestration dealing with the allocation of resources to support the VNFs and the logical links. In the context of METRO-HAUL, a network slice consists of a Network Service (NS) deployed using the NFVO spanning multiple nodes and network domains; the VNF placement functionalities are provided by the Network Planner. The Virtual Infrastructure Manager (VIM) is the responsible for the management of the NFV Infrastructure (NFVI) and the instantiation of the Virtual Machines (VM) of the VNFs in a single Datacenter (DC) domain (AMEN/MCEN).

2) The *WAN Infrastructure Manager* (WIM) is used by the NFVO to orchestrate network resources and it is responsible for the provisioning of connectivity paths between VNFs. The WIM architecture is hierarchical and aligned with IETF ACTN [ACTN], with an SDN control per technology domain. Running on top of the SDN hierarchy, the parent SDN controller abstracts the underlying complexity and presents virtualized networks to their customers.

3) The *Monitoring and Data Analytics* (MDA) [Ve18.2], responsible for implementing autonomic networking. The monitoring system has the capability to do measurements on the data plane and to generate data records that are collected and analyzed by the MDA subsystem to discover patterns (knowledge) from the data. In METRO-HAUL, the MDA is distributed [APV17] and consists of MDA agents that run in the network nodes and are responsible for monitoring data collection, aggregation, and knowledge usage. Aggregated monitoring data is conveyed to the MDA controller. From collection, data can feed ML algorithms (see Section 2.2), which can be used to issue re-configuration/re-optimization recommendations towards COM modules, such as an SDN controller or orchestrator.

4) The Placement, Planning, and Reconfiguration Subsystem (*Network Planner*), responsible for optimizing the resource allocation to effectively provision services featured by heterogeneous requirements and for applying different policies and strategies. This task comprises the provisioning of VNFs in specific METRO-HAUL nodes, and the allocation of network resources.

### 2.1.3 Multi-domain Networks

Single operators' transport networks are usually created as multi-domain networks a result of deploying nodes from different vendors and/or different technologies. In such scenarios, the topology of the different domains is fully visible from outside each domain and therefore it is possible to compute end-to-end paths using one single or a set of coordinated SDN controllers.

In contrast, as a result of privacy policies, in multi-operator multi-domain networks only an abstraction of the topologies is visible from outside the domain, which prevents from computing paths traversing more than one domain.

Recent works (e.g., [Ca16]) proposed using *market-driven brokers* on top of SDN controllers in charge of each domain. That scheme provides autonomy to the domains while improving scalability. Under this approach, a multi-operator network, is the result of connecting single-operator networks, each with an SDN controller, and a broker layer coordinating end-to-end multi-operator provisioning on top of the domains (see Fig. 2-2).

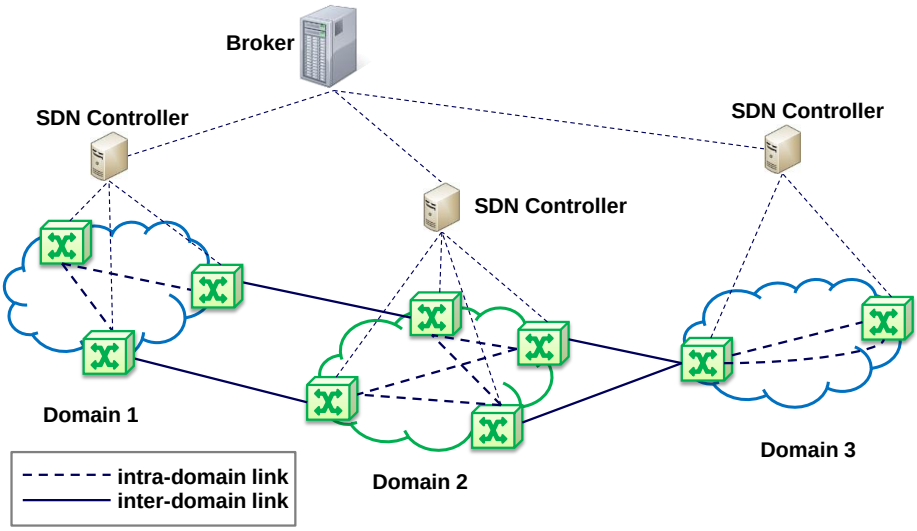


Fig. 2-2 Multi-operator network architecture [Ca16]

When a multi-operator path computation is requested, the broker collects intra-AS abstracted connectivity, bandwidth availability and other performance metrics from the inter-domain SDN controllers. Observe that, each SDN controller advertises an abstracted intra-domain link information to the broker that depends on both, internal domain policies and the specific agreement with the broker. Details of the intra-domain topology remains concealed from the rest domains and the broker.

## 2.2 Machine learning algorithms

ML is typically thought of as a universal toolbox, ready to be used for *classification*, i.e., identifying to which of a set of categories a new observation belongs to, and *regression* i.e. estimating the relationships among variables; this is sometimes generically referred to as knowledge discovery in databases. When, in fact, it is a diverse field comprising of various constituents and necessitates a software ecosystem including data collection and transformation, model selection and optimization, performance evaluation, visualization, model integration, to name a few. Explicitly, ML refers to computational representation of a phenomenon, aiming at execution of a task, given a certain performance, based on a given environment.

ML approaches may be categorized based on objectives of the learning task, where these objectives may target pattern identification for classification and prediction, learning for action, or inductive learning methods. The algorithms may be further classified into three distinct learning families [Ma11], *supervised learning*, *unsupervised learning* and *reinforcement learning (RL)* (see Fig. 2-3). Semi-supervised learning -or hybrid learning- is sometimes considered as a fourth family, borrowing features from supervised and unsupervised ones [Cha10].

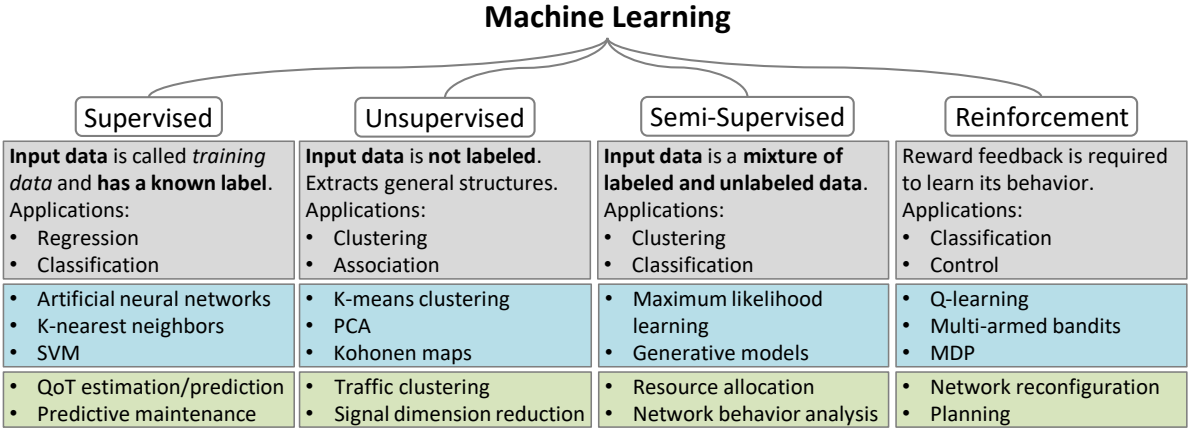


Fig. 2-3. ML Families (reproduced from [Ra18]).

We introduce next the ML algorithms relevant for this Ph.D. thesis, which belong to the supervised learning (SL) family. SL makes use of known output feature(s), named *labels*, to derive a computational relationship between input and output data. An algorithm iteratively constructs a ML model by updating its weights, based on the mapping of a set of inputs to their corresponding output features. SL may be further categorized into classification and regression tasks, depending on whether discrete or continuous output features are used [Ra18].

### 2.2.1 Support Vector Machine

Support Vector Machine (SVM) is a classification technique targeted at maximizing margins for samples of different classes and could be categorized into linear and non-linear SVM by a decision boundary called hyperplane. In linear SVMs, the input data are linearly categorized (Fig. 2-4a), whereas in the non-linear case, the inputs are transformed into another space by a kernel mapping (Fig. 2-4b).

A common kernel mapping is Gaussian radial basis function:

$$k(x_i, x_j) = \exp(-\gamma . |x_i - x_j|^2), \tag{2-1}$$

where $\gamma > 0$, $x_i$ and $x_j$ are two samples.

The algorithm is also classified by soft and hard margins. In a binary classification case, the margins are considered hard when they are presented by 1 and -1. When the following loss function is maximized, the margins are considered as *soft*.

$$\max(0,1 - y_i(\omega \times x_i - b)) \tag{2-2}$$

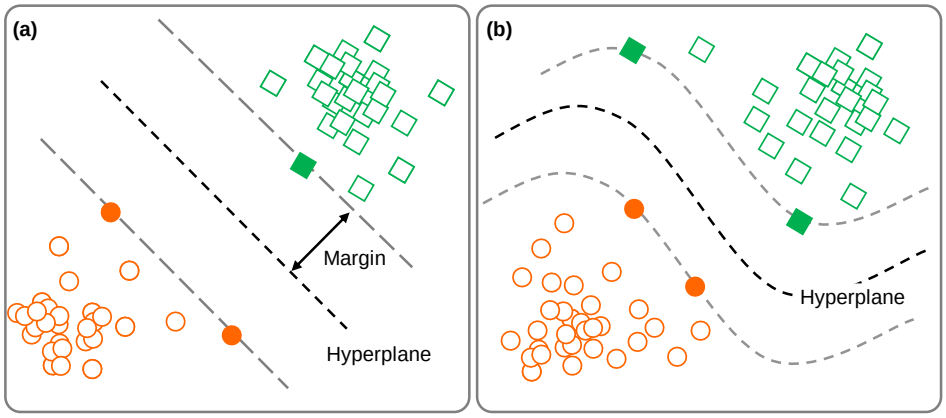


Fig. 2-4. Classification problem with a) linear model and b) non-linear model.

Where $w$ represents the weights and $b$ is the biases of connection.

### 2.2.2 Artificial Neural Networks

A Feedforward Artificial Neural Networks (FFNN) is a ML approach comprising of one input and one output layer, and one or more hidden layers in-between; where each layer depends on use cases could be composed of several neurons. Features ($X$) are fed into the network though the input layer, where the neurons in the input layer are connected with the neurons in the next layer, i.e. the first hidden layer, and so on. The neurons on the output layer are directly connected with the outputs $Y$ (see Fig. 2-5).

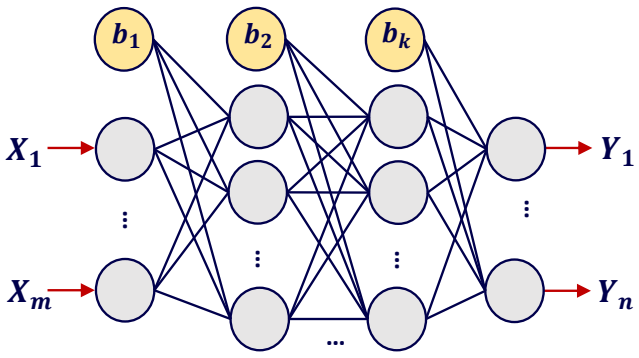


Fig. 2-5 A scheme of a FFNN with input features and output responses

In a typical fully connected FFNN, the nonlinear mapping between layers could be given by:

$$relu(X + W + b) \tag{2-3}$$

where $W$ represents the weights, $b$ the biases of the connections, and $relu$ is the activation function as:

$$\max(0, x) \tag{2-4}$$

Further FFNN parameter optimizations are carried out using algorithms like gradient descent, etc.

The use of FFNN allows considering complex nonlinear relations among input features and the predicted future event. Moreover, they facilitate working with a mix of numerical and categorical inputs, as well as making predictions for several steps ahead, i.e., multi-step prediction.

Although FFNN can be both designed and trained to predict time series events, they fit better for applications that do not depend on time. On the contrary, Recurrent Neural Networks (RNN) [Ma02] have been proposed specifically to deal with time series events, since they can explicitly manage the ordering among inputs. RNNs implement knowledge persistence, so it can be used for predictions. However, in general, this memory is short and knowledge vanishes with time.

### 2.2.3 Long Short-Term Memory

To improve RNNs, Long Short-Term Memory (LSTM) networks were proposed to expand temporal dependence learning. LSTM units consist of a set of different complex gates, namely *input*, *output*, and *forget* gates and the coefficients of the network are dynamically managed to keep long term memory. LSTMs provide accurate prediction of time series with complex temporal correlation, e.g., periodical sharp changes.

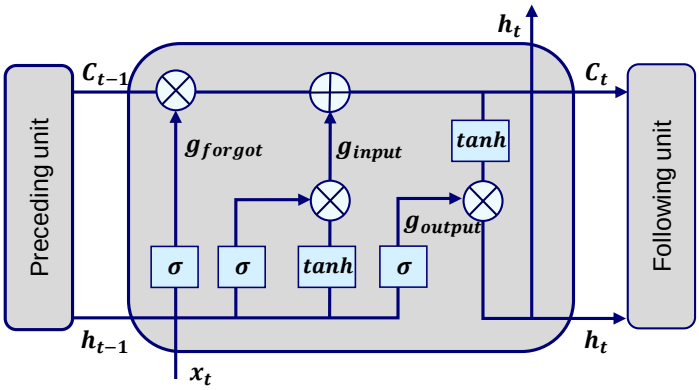


Fig. 2-6. LSTM cell structure

A general structure of an LSTM unit is depicted in Fig. 2-6 to forward internal coefficient, memory ($C_{t-1}$) and hidden ($h_{t-1}$), from earlier to next unit.

To better understand the unit mechanism, let us start with the primary *forgot gate* insert the input and last hidden state into a sigmoid function that is formulated as:

$$g_{forget} = \sigma(W_{forget}(x_t + h_{t-1}) + b) \tag{2-5}$$

Where $W$ denotes the weight of each inputs of *forget gate* and $x_t$ is current information; the output value is between 0 and 1 to discard or keep $C_{t-1}$ information. In order to update $C_{t-1}$, *input gate* employs a sigmoid and a tangent hyperbolic that response between -1 and 1, as following:

$$g_{input} = \sigma\big(W_{input}(x_t + h_{t-1}) + b\big) * \ tanh(W_{input}(x_t + h_{t-1})) \tag{2-6}$$

Therefore, the memory $C_t$ calculated by means of response from forgot and input gates:

$$C_t = g_{forgot} + g_{input} \tag{2-7}$$

Finally, the regulated output gate and hidden state from this cell can be written as:

$$g_{output} = \sigma(W_{output}(x_t + h_{t-1}) + b) \tag{2-8}$$

$$h_t = g_{output} + tanh(C_t) \tag{2-9}$$

### 2.2.4 Optimization for training ML model

*Mathematical programming* or *optimization* is a mathematical method to find an *optimal* point $x^*$ that results into the minimum (or maximum) value of a function $f(x)$ while satisfying a set of constraints [Ch83]; such point $x^*$ is said to be optimum. More formally, an optimization problem can be defined as follows.

$$\min \quad z = f(x) \tag{2-10}$$

subject to:

$$g_i(x) \geq b_i \quad \forall i \in R \tag{2-11}$$

where $x$ represents a vector of variables, $f(x)$ is the objective function, and $R$ represents the set of constraints. A constraint is an inequality defined by a function $g_i(x)$ and a constant $b_i$.

Let us define $X$ as the set of all possible $x$ vectors. Then, we can define the set $S$ of feasible solutions of the problem, as follows:

$$S = \{x' \in X | g_i(x') \geq b_i\} \quad \forall i \in R \tag{2-12}$$

i.e., $S$ contains all elements in $X$ satisfying the whole set of constraints. Note that a problem could have alternative optimal solutions, i.e., several $x^*$ with the same $z^*$ value. Therefore, we can define the set of optimal solutions $X^*$ as:

$$X^* = \{x' \in S | f(x') \leq f(x''), \forall x'' \in S\} \quad (2\text{-}13)$$

The problem, however, could have no feasible solution, i.e., $S = \emptyset$; in such case the problem is unfeasible. Finally, when $f(x^*) = -\infty$ the problem is unbounded.

A *Linear Programming* (LP) problem is a special case of mathematical programming, where $f(x)$ and $g_i(x)$ are linear functions of real variables. When variables are restricted to be integer, the problem is called *Integer Linear Programming* (ILP), whereas if the problem combines integer and real variables, the problem is defined as *Mixed Integer Linear Programming* (MILP). Finally, *Non-Linear Programming* (NLP) entails, at least, one non-linear function [Ch83].

Optimization is exploited in ML for training purposes so as to find the values of the hyperparameters of the model that minimize the error [Su20]. In this regard, Gradient Descent is a well-known iterative optimization algorithm that can be used for finding local minima of a given function.

## 2.3 Generation of reliable and accurate synthetic data

How to gather data for training ML algorithms is one of the main challenges that need to be solved. The objectives to be achieved include not only the quality of such dataset, which is directly related to the final accuracy of the prediction for network operation, but also the time needed for that collection. Note that in many cases, performance-related data heavily depends on the actual characteristics of the network entity of interest and are only available when such entity is set-up. For instance, QoS measurements depend on the actual routing of a connection; in consequence, real measurements can only be available after such connection is established, and might change due to the provisioning of neighboring connections. However, ML algorithms need to be ready to be deployed at connection set-up time and thus, special techniques are needed to train accurate ML algorithms before data for that specific network entity is available. Further, the inherent prediction ability of ML algorithms can be used during the lifetime of the network entity to elastically allocate resources to the optimality.

Synthetic data generation is one of the solutions that can be implemented for the identified challenges and run in a sandbox domain. However, for the generated data to be reliable and accurate, they must be generated using techniques that rigorously reproduce the real scenario, thus creating a digital twin. Such a digital twin can be based on a combination of analytics and simulation models, which need to be tuned

using the characteristics of the real entity, as well as with real measurements collected before or during operation.

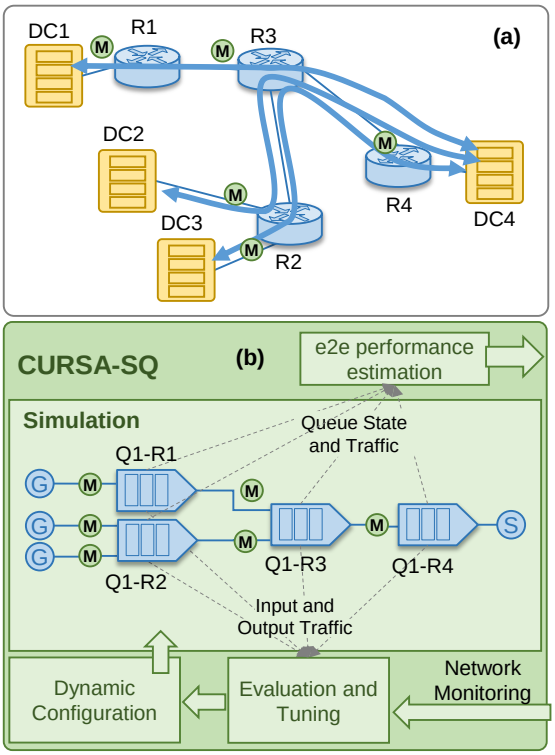


Fig. 2-7. Example of digital twins for the packet (a-b).

To illustrate accurate data generation, Fig. 2-7 presents an example of a digital twins for the packet layer. Fig. 2-7a presents an example of a network with four nodes interconnecting four DCs, where DC1-3 exchange data with DC4 (flows are also represented). Let us assume that traffic is monitored at the input interfaces, so a number of observation points have been activated. A digital twin is represented in Fig. 2-7b based on the CURSA-SQ methodology [Ru18]. CURSA-SQ includes a continuous G/G/1/k queue model with a first-in-first-out discipline based on the logistic function, which enables solving the model in near-real time. To accurately reproduce the real scenario, however, parameter tuning for the queues is required. To that end, the *dynamic configuration* module is in charge of defining the traffic to be generated and consumed by every DC, as well as the entities configuration, which evaluates the accuracy of the estimation by comparing it against the real traffic conditions measured from the observation points in the network.

## 2.4 Conclusions

The purpose of this chapter has been giving the needed background to ease the comprehension of some concepts used in this Ph.D. thesis. Starting from a quick

survey of the infrastructure and control plane architecture assumed in this Ph.D. thesis, main ML algorithms used on the next chapters are summarized. Finally, ideas related to data generation, which is a key issue for developing and testing ML algorithms, have been introduced.

# Chapter 3

# State-of-the-Art

Autonomous network operation reduces human intervention related to the configuration of the network. Such operation requires collecting performance measurements from the network and developing intelligent algorithms that make decisions proactively to reach some performance defined for each network entity. A related concept is that of independent operation vs coordinated operation. Independent operation occurs when the decisions that are made on a network entity are based on measurements collected for the same entity. However, since in a network infrastructure many entities are sharing the set of common resources, pure independent operation is rare, as it can lead to overall suboptimal resource utilization and even to result in poor performance because of the natural competence for resources. Therefore, some kind of coordination among entities should be devised.

In this chapter, we present a review of the state-of-the-art related to coordination in framework of network management, among ML-based entities. The state-of-the art related to each goal defined for this Ph.D. thesis is review 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 this research work.

## 3.1 KM in single domain networks

Several works in the literature have focused on implementing autonomous control loops entailing knowledge usage and decision making.

The authors in [Ru19] present a predictive Autonomic Transmission Agent (ATA) based on ANN that predicts the right Forward Error Correction (FEC) algorithm configuration for short-term operation as a function of real-time monitoring of state of polarization (SOP) traces and the corresponding pre-FEC Bit Error Rate (BER).

Note that the control loop is performed at the device level, and so the knowledge usage and the decision-making process.

The authors in [Sh19] explore several ML approaches based on SVMs for fault management, specifically for soft-failure detection, identification and localization. Note this is a distributed system where knowledge usage is placed at the device level and decision-making is placed close to the centralized SDN controller. The authors in [Ve18.1] demonstrate the concept of autonomic networking in disaggregated scenarios through use cases for provisioning and self-tuning based on the monitoring of optical spectrum. Note that here the control loop entails collecting monitoring data from one device and tuning the configuration of another one, so knowledge usage and decision-making need to be placed in some centralized element. Finally, the authors in [Mo17.2] model origin-destination (OD) traffic at the packet layer and use the traffic prediction to proactively reconfigure the virtual network topology (VNT) to adapt it to current and predicted traffic volume and direction. Note this is a purely centralized autonomous networking control loop case where knowledge usage and decision-making are placed in close to the centralized SDN controller.

The authors in [Ve19.2] present the benefits of adding a MDA system and present operators use cases looking at automating optical network operation. Several MDA architectures are overviewed, from the centralized to alternative hierarchical ones that allow to implement control loops at different levels. In addition, the works in [Gi18] and [Ve18.2] provide more details regarding MDA architectures and its integration with other elements in the control and data planes.

Instead of data, ML models can also be shared among agents. An example of such model sharing can be found in [Mo17.1], where the authors proposed to model OD traffic in the core as an aggregation model of the conveyed metro flows models. In this case, metro flow models are trained by the metro SDN controllers and shared with the core SDN controller, which composes the model for the core OD.

To the best of our knowledge, no works in the literature have focused on providing structure to collective self-learning in the context of networking. The KM process entails creating and sharing knowledge. In particular, in this Ph.D. thesis, we target at completing the KM process for truly autonomous optical network operation; we apply KM in the context of optical transmission and networking and define it as the process to autonomously (i.e., without human intervention) *i*) *discover*; *ii*) *share*; *iii*) *assimilate*; and *iv*) *use* knowledge to improve the performance of a network. Note that networks consist of a set of networking devices, which would probably not achieve a global improvement in case of knowledge being individually managed.

## 3.2 KM for multi-layer networks

Solutions currently under research to guarantee the requested performance are Network Function Virtualization (NFV) and network slicing, where NFV Network Services (NS) consist of interconnected Virtual Network Functions (VNFs) placed in different COs. Note that, as specific network resources are reserved to every NFV NS, the performance is guaranteed at the cost of high overprovisioning unless dimensioning is carefully carried out. Even though the performance is bounded, it cannot be precisely estimated as a function of the input traffic, which might be of interest for both network operators to reduce overprovisioning, and for customers to implement autonomic NFV services (see, e.g., [Ve18.1], [Ve19.3], [Ru16.1]).

The performance of a layer 2 (L2) / layer 3 (L3) packet connection can be assessed during the commissioning phase through active monitoring, as we demonstrated in [Lo19], using a 100 Gb/s active probe. We measure a packet connection by using an active probe at the source to inject a train of numbered and timestamped packets; when the train arrives to the other end of the connection, another active probe measures throughput, by using the reception times of every packet, and latency, by comparing the transmission timestamp with the reception time of each packet. Note that this latency measurement requires a common reference clock for the active probes, which is provided by a Global Positioning System (GPS) receiver to achieve the needed accuracy [MR16.1]. In the case that the connectivity is implemented by a point-to-multipoint (p2mp) multicast connection [Ru15], instead of a point-to-point (p2p) one, every probe in a destination will measure the performance. Following this procedure, packet connection performance, i.e., one-way packet delay, delay variation (jitter), packet loss, and throughput, can be measured (see [MR16.2], [Le18] for details). However, as the length of each measurement train and the packet separation are constant, the obtained measurements can be considered as a bound, since they are not related to the specific traffic that the connection will support.

ML models can be trained and used to estimate the performance of packet connections. However, to obtain accurate models, training and validation procedures need to be carried out, which entails the availability of a large amount of data. Obtaining specific data to model a given connection takes a long time and as NFV NSs might be highly dynamic, a different approach is required to reduce the time to create the training and testing datasets.

To the best of our knowledge, no works in the literature have focused neither on KM for multilayer networks nor a methodology to recreate the real conditions on which the connections will be working. In this Ph.D. thesis, we help to improve the predictability, as well as to assess the performance of connectivity services; ML models (e.g., feedforward ANN) can be trained and used to estimate the performance of end-to-end packet connections. Note that by considering ML models for such estimation, the details of the network are abstracted and thus, they can be shared with the final customers. However, to obtain accurate models, training and

validation procedures need to be carried out, which entails the availability of a large amount of data.

## 3.3 KM for broker-based multi-domain networks

The provisioning of client end-to-end connectivity services across heterogeneous multi-operator (Multi-AS) networks is a challenging task. Further, in the current context of new applications and services, the provisioning of end-to-end connectivity cannot be only focused on ensuring throughput; the delay needs to be bounded to ensure its right operation. Therefore, the broker must be in charge of determining the route of the end-to-end connections across the individual domains so as to ensure that the required QoS is met at the set-up time, as well as during the connection life-time. QoS should be thus monitored periodically by collecting counters from routers (passive monitoring) or by using active probes (active monitoring). In the latter, two probes can measure the round-trip time by injecting trains of numbered and timestamped packets that are looped back by the remote probe [Lo19].

Aiming at collecting fine grain measurements, active monitoring can be used, which consist on injecting packet trains and measuring data were used afterwards to produce a specific delay model for the packet connection that could be used during connection operation time. However, using active probes requires dealing with the introduced overhead, and they are mostly used during commissioning testing. Aiming at collecting fine grain measurements, In-band Network telemetry (INT) can be adopted to collect network status hop-by-hop, which fits well in single operator SDN environments. However, precisely the hop-by-hop characteristics of INT difficulties its application on multiple operator scenarios, as it might reveal internal details of the domains. Another option is network tomography [He21], which consists in inferring domain and inter-domain link delay components from e2e measurements together with the available topology and routing information. Nonetheless, precisely topology and routing information, is not generally available in multi-operator scenarios.

Regarding the application of ML on the collected monitoring data, some previous works have proposed models for short-term and long-term traffic prediction at different time scales (see, e.g., [Ni20] and [Mo17.2] for second and day time scale, respectively). ML can be also used to predict the delay, which should be bounded by the maximum delay defined at the provisioning time. For instance, ANNs are proposed in [Kr20] to model e2e delay, where the authors used traffic matrices (generated with the NS-3 simulator assuming Gaussian distributed traffic intensity for each flow) as input of the predictive models and evaluated their robustness and accuracy under different network scenarios. Other works have proposed alternative networking models together with control and orchestration plane architectures to provide the committed delay to customer connections. The authors in [Rk20]

leveraged a single domain SDN paradigm and proposed a model able to relate the complex network relationships to produce accurate estimates of the per-packet delay distribution and loss. The authors in [Za21] proposed a network slicing orchestration solution able to handle e2e latency in multidomain single-operator networks. They leveraged a multi-armed-bandit method to allocate resources to slices to meet end-to-end latency requirements. Current networks are non-stationary in general and therefore, pre-trained networking models require fine-tuning to correct the model-mismatch problem, as highlighted in [Do20].

Multi-operator networks bring additional challenges, in particular regarding e2e delay. One possible architecture to provide e2e services is that of peer-to-peer, where operators exchange information among them directly. As an example, the authors in [So20] studied the convenience of exchanging information related to the guaranteed latency and resource availability in each of the domains to reduce service provisioning blocking probability.

Instead of peer-to-peer multidomain architectures, an e2e service provider can deploy a broker system (e.g., based on the one proposed in [Ca16]) that coordinates e2e path provisioning and relies on domain SDN controllers for intra-domain provisioning. Here, domains would share information with the broker in targeting at providing better services while receiving performance feedback from the broker. Nonetheless, for that sharing to be realistic, exchanged information needs to be conveniently abstracted, so to ensure that internal details of the domain are not revealed. In that regard, the authors in [Ch18] proposed a knowledge-based multidomain service provisioning framework, where intra-domain topologies are *abstracted* and shared with the broker for multidomain network automation tasks. A similar approach was proposed by the authors in [Pa16], where abstracted topology was used for inter-domain connection provisioning and QoS assurance.

Taking advantage of abstracted information is not always an easy task and it might lead to poor adaptability and resource efficiency. In this context, some previous works have proposed to leverage ML to develop cognitive multidomain provisioning schemes. The authors in [Ch19.1], formulated the operations of multidomain networks as a multi-agent learning system and presented a multi-agent Deep RL approach to enable domain controllers and brokers to learn cooperative provisioning policies from performed operation experiences. The authors in [Zh19] proposed an ML model to produce inter-domain routing solutions autonomously by taking advantage of historical provisioning traces. Finally, the authors in [Ch19.2] demonstrated collaborative learning schemes for accurate quality of transmission estimation in multidomain optical networks, where the distributed ML blocks deployed in the broker and domain controllers learn inter-domain QoT estimators by exchanging just necessary learning data.

To the best of our knowledge, no works in the literature have focused on the coordination of ML capabilities between domain network controllers and the broker.

In this Ph.D. thesis, we address these issues and proposed a collaborative environment based on sharing delay models.

## 3.4 Traffic prediction and KM in Intent-based networking

The traffic generated by some services might be complex and hard to model due to the presence of multiple periodicities ranging from few hours to several days. Although 24h is still the dominant periodicity for most services, other periodicities with shorter or larger period can introduce sharp traffic changes that significantly distort the typical daily profile. This fact makes impractical applying traditional predictive approaches in many multilayer optical network automation scenarios, e.g., dynamically allocating capacity to a traffic flow according to traffic prediction [Mo17.1]. Aiming at modelling complex time series, such as network traffic, LSTM can be an option due to its ability to learn data with long-term sequential dependencies and indefinite duration.

In this regard, authors in [Bi21] proposed an integrated model based on an LSTM predictor that filters noise from data and predicts timeseries with long-term view and significantly reduces prediction error. Another approach can be found in [Tr18], where authors proposed a multi-step LSTM-based predictor for traffic data in a physical channel with temporal characteristics. Results compared with ARIMA and FFNN show lower prediction error.

Together with LSTMs, specific loss functions have been recently proposed to minimize prediction error in the presence of sharp changes that provides accurate prediction of time series with complex temporal correlation, e.g., periodical sharp changes [Gu19]. Nevertheless, although the overall accuracy of LSTMs can be higher than that of alternative traffic prediction methods, they still incur in occasional errors that can reduce robustness for autonomous network operation, especially when the scale of changes is high, where traffic under-prediction can lead to connection capacity under-provisioning, which translates into traffic loss.

To the best of our knowledge, no works in the literature have combined LSTM and RL for capacity allocation based on traffic prediction. In this PhD thesis, we combine LSTM-based traffic prediction model and RL-based capacity allocation.

## 3.5 Conclusions

In this chapter, we have reviewed the state-of-the-art of relevant works related to the goals of this thesis. Table 3-1 summarizes the study.

*Table 3-1: State-of-the-art summary*

| Goals | References |
|---|---|
| **KM in single domain networks** | [Ru19], [Sh19], [Ve18.1], [Mo17.2], [Ve19.2], [Gi18], [Ve18.2] |
| **KM for multi-layer networks and Intent-based Networking** | [Ve18.1], [Ve19.3], [Ru16.1], [Lo19], [MR16.1], [Ru15], [MR16.2], [Le18] |
| **KM for broker-based multi-domain networks** | [Lo19], [He21], [Ni20], [Mo17.2], [Kr20], [Rk20], [Za21], [Do20], [So20], [Ca16], [Ch18], [Pa16], [Zh19], [Ch19.2] |
| **KM in Intent-based networking** | [Mo17.1], [Bi21], [Tr18], [Gu19] |

In view of this study, we can conclude that, although some previous works have proposed algorithms and different methods for autonomic network management, they have not considered in-deep scenarios with multiple entities that need to exchange data and models.

In this and the previous chapters we have reviewed the state-of-the-art and the background concepts needed to fully understand this work. The following chapters present the essence and contributions of this Ph.D. thesis.

# Chapter 4

# KM Architecture, Methods and Use Cases

Autonomous network operation realized by means of control loops, where prediction from ML models is used as input to proactively reconfigure individual optical devices or the whole optical network, has been recently proposed to minimize human intervention. A general issue in this approach is the limited accuracy of ML models due to the lack of real data for training the models. Although the training dataset can be complemented with data from lab experiments and simulation, it is probable that once in operation, events not considered during the training phase appear thus leading into model inaccuracies. A feasible solution is to implement self-learning approaches, where model inaccuracies are used to re-train the models in the field and to spread such data for training models being used for devices of the same type in other nodes in the network.

In this chapter, we develop the concept of collective self-learning aiming at improving models error convergence time, as well as at minimizing the amount of data being shared and stored. To this end, we propose a KM process and an architecture to support it. Besides knowledge usage, the KM process entails knowledge discovery, knowledge sharing, and knowledge assimilation. Specifically, knowledge sharing and assimilation are based on distributing and combining ML models, so specific methods are proposed for combining models. Two use cases are used to evaluate the proposed KM architecture and methods. Exhaustive simulation results show that model-based KM provides the best error convergence time with reduced data being shared.

# 4.1 KM in optical networks

## 4.1.1 KM Process Overview

Fig. 4-1 presents the architecture proposed to enable KM, where two software agents in charge of networking devices are represented. Agents collect monitoring/telemetry data from the underlying device(s) e.g., an optical transponder (step 1 in Fig. 4-1a) that are consumed by a ML-based application, to produce some output (e.g., prediction) based on some ML models regarding some device/entity, e.g., the QoT of an optical connection. The results can be used by a decision maker module (2) to tune configuration parameters in the device(s) (3). Note that we just described the typical control loop (1-2-3), which focuses exclusively on *knowledge usage*.

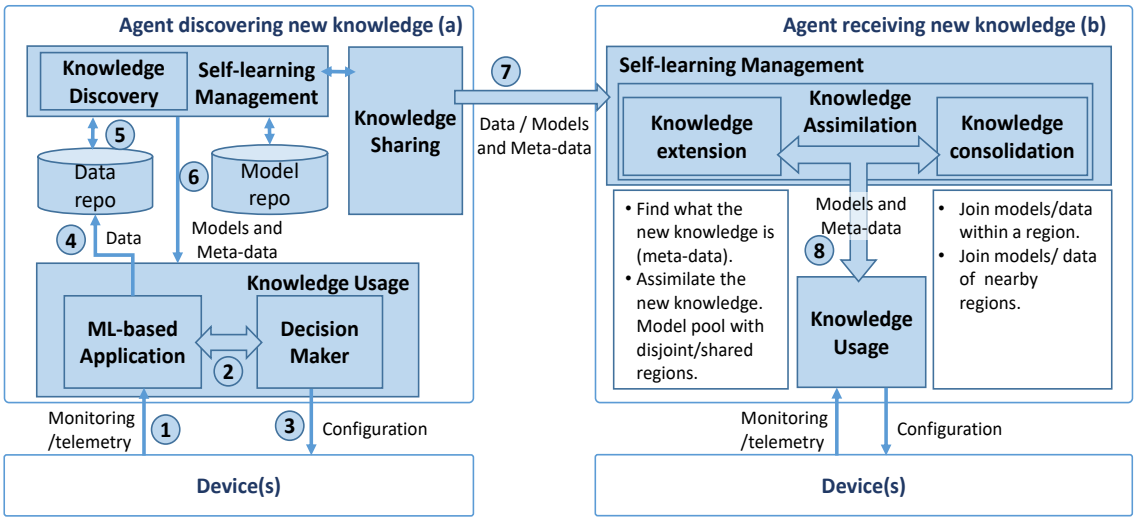


Fig. 4-1. KM Process. New knowledge is discovered (a) and assimilated for operation (b).

Now let us assume that the output produced by the ML-based application based on the measured data is stored (4) and that such output could be compared to real data measured from the device(s) after some time. If this would be possible, we could conceive an algorithm that would monitor the accuracy of the current ML models and detect events for which the models return inaccurate output (5). For illustrative purposes, Fig. 4-2a shows an example where a model for regression has been trained with data points. Note that those data points do not need to be uniformly distributed in the regions and can form data clusters in some regions of the *features space*, whereas no data points can be found in other regions. A prediction for data in an unknown region would produce a response value that might be far from the actual response measured from the network. Thus, detecting such inaccuracies would open the opportunity to increase our training dataset with new labeled data (i.e., $<X, y>$, where $X$ is the input data and $y$ the predicted response) and apply ML training to produce more accurate ML models that can be immediately used by the ML-based

application (6). This loop (4-5-6) entails *knowledge discovery* and it is the base for *self-learning* [Ve19.1].

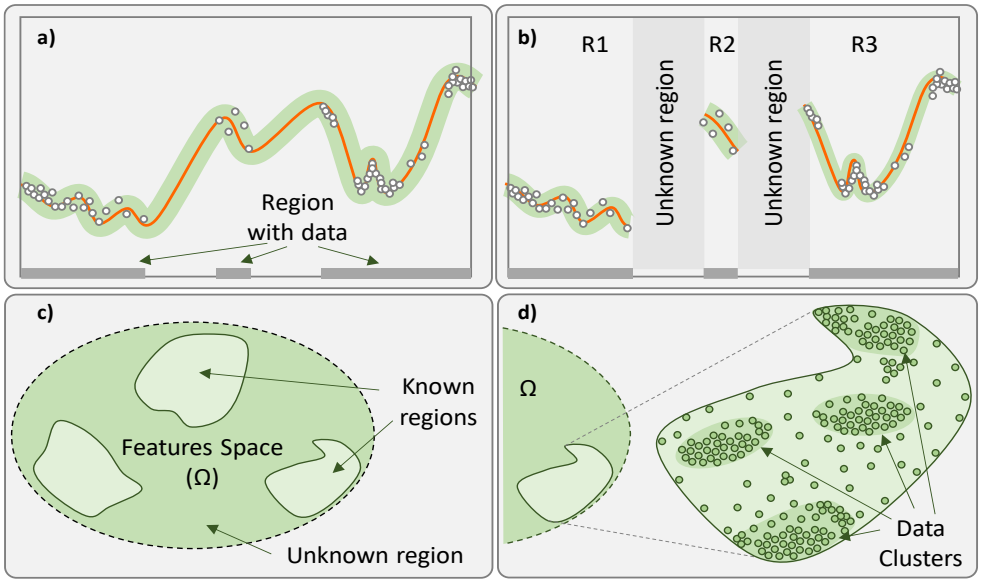


Fig. 4-2. Known and unknown regions in the features space.

As an alternative to the single ML model covering the complete features space, one could analyze the structure of the training dataset and realize of the presence of data clusters. In such case, specific and more accurate ML models could be produced within each of the selected regions as it is suggested in the example in Fig. 4-2b (regions R1..3). In this case, some information (*meta-data*) is needed to specify the region of applicability of the model, as well as other important data, like the number of samples used to produce the model, etc. In addition, note that the lack of a model in the region of a collected measurement reveals a new unknown region; those collected data need to be stored until the corresponding label is obtained and can be used to extend the knowledge to that region.

Imagine now that the knowledge discovery process is performed individually per every different device/entity, as the measured data could be specific for such device/entity and so the corresponding ML models. In such case, knowledge discovered from one device/entity cannot be shared among different devices/entities. However, let us assume that either the measured data can be used unchanged by other devices/entities or there exists a function that normalizes the measured data (i.e., removes local dependences) so that the resulting normalized data can be used to train ML models for other devices/entities. Then, new *knowledge* in the form of labeled data can be *shared* with other agents as soon as it is discovered (7), thus enabling *collective learning* [Ve19.1]. Note that the normalized data received from other agents can be used to complement the local training dataset; this increases the learning speed since the probability of rare events to be observed increases as there are more observers.

However, sharing knowledge in the form of labeled data might entail the exchange of large volumes until the accuracy of the ML models does not reach high values. Note that one single labeled data point consists of a tuple of values and that a complete training dataset can contain a large amount of data points. Another alternative to reduce the amount of data being exchanged is to produce specific models for the knowledge just discovered. These models can be very accurate in a particular region of the features space where the new knowledge has been discovered.

The components related to KM in the agent receiving the new knowledge are sketched in Fig. 4-1b. Note that the separation between the agent receiving the new knowledge and the one discovering it is done for illustrative purposes, as there is no limitation about being actually the same agent.

When a model and meta-data are used to share new knowledge, the receiving agent needs to *assimilate* such knowledge, starting by understanding what the new knowledge is. Assuming that the feature space is modeled in a per-region way, the received knowledge can be located (totally or partially) in one or more of the known regions or in the unknown region; in the former, the model is added to the found region(s) and a merge of regions could be performed, whereas in the latter, a new region is created. We name *knowledge extension* to the process of identifying the new knowledge and updating the regions. Note that a region can be modelled using one or more models, so region updating would entail generating a new model joining the previous model with the received one, or just adding the new model to the pool of models. Another process that we call *knowledge consolidation* is in charge of joining models within a region and joining nearby regions. Fig. 4-2c-d illustrate the features space of a given problem, where the training dataset contains labeled data grouped into three different regions. However, data points are not usually uniformly distributed along a region, as regions are dynamically re-defined as a result of a region merging process, triggered whenever new knowledge arrives. Finally, changes in the regions and models and meta-data generate new *operational* models that are ready for knowledge usage (step 8 in Fig. 4-1b).

### 4.1.2 Proposed Architecture

Fig. 4-3 presents an extended architecture for KM, where more details of the agent are depicted; specifically, *knowledge discovery and knowledge assimilation* in the form of extension and consolidation (collectively named *self-learning*), *knowledge sharing*, and *knowledge usage* components are detailed. In addition, the *Knowledge Manager* component coordinates KM operations.

The data collected from the underlying physical device(s) is processed by an *application manager* that uses knowledge for the autonomous control of the device(s). For the sake of generalization, we consider that the configuration of the devices is based on a set of algorithms for different problems, which generate outputs

to a decision maker module in charge of finding the best configuration for the forecasted conditions. Any problem might require a specific procedure combining several techniques (ML, statistics or mathematics) to generate its outputs. The role of the application manager in the device control loop is to feed the different problems with the required inputs and to adjust the decision maker according to the observed local performance.

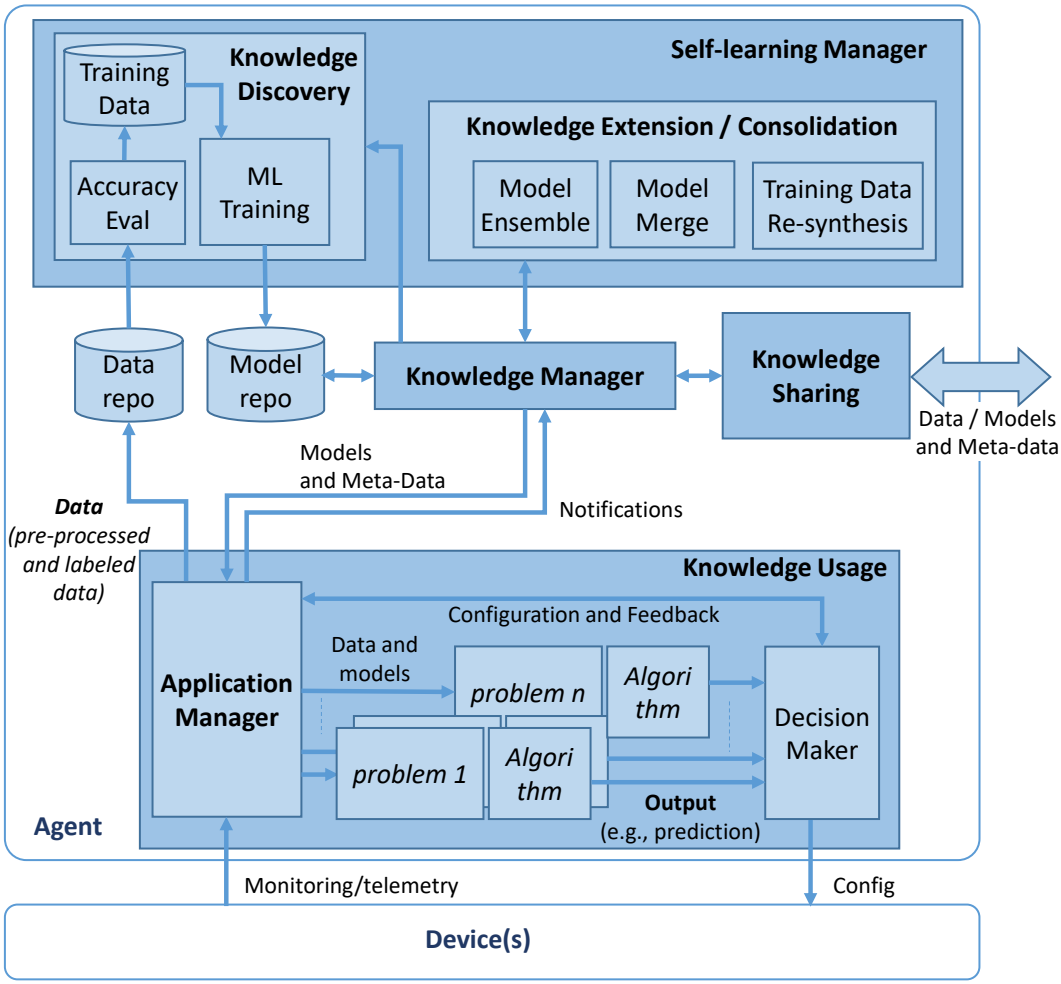


Fig. 4-3. Detailed architecture for KM

In addition to these operational tasks, the application manager exports pre-processed and labeled data (including model predictions and real measurements) to be stored in the *data repository*. Such data is analyzed by the knowledge discovery module, which holds two essential roles: *i*) to identify inaccuracies in the current ML models and, *ii*) to populate its internal training dataset and perform ML training to produce new models that are stored in the *model repository*.

The knowledge discovery loop is the main source of knowledge acquisition coming from real data from the operation of the underlying device(s). Such new knowledge can be afterwards shared with other agents through the knowledge sharing module

thus, implementing collective self-learning. Consequently, knowledge discovered by other agents is also received and stored in the model repository.

The activity of knowledge discovery could lead to many ML models being stored in the repository, which would hinder knowledge usage. For example, in the case of keeping several ML models restricted to narrow region in the feature space or alternatives models for the same region. Owing to that fact, knowledge assimilation applies methods for knowledge extension and consolidation focused on reducing the number of models used for operation while keeping its overall accuracy. As illustrated in Fig. 4-3, we consider three different methods for such task, named *model ensemble*, *model merge*, and *training data re-synthesis*. The next section is devoted to providing the details for these assimilation methods. Finally, following a given scheduling policy, e.g., every time a new ML model is made available or with some periodicity, the knowledge manager updates the ML models of every problem in the knowledge usage module, so the algorithms can use them for operational purposes.

Last but not least, the knowledge usage module plays a pro-active role to speed-up knowledge discovery, as the algorithm can discover that some given measured data locates into an unknown region of the features space of their problems. In such case, the application manager notifies the knowledge manager, which requests the knowledge sharing module to ask other agents about labeled data around the measured one, so as to produce a specific ML model for that unknown region.

## 4.2 Knowledge Assimilation

In this section, we describe in detail three elementary methods for assimilating knowledge in the previously described context. These options, presented in Fig. 4-4, are used for knowledge extension and consolidation. For the sake of simplicity, let us assume that the agents focus on one single problem and that they are prepared to perform all type of modelling procedures including self-supervised learning. Regarding the typology of problems, let us consider both classification and regression ML-based applications; due to their properties, we selected SVM for classification and ANN for regression.

Without loss of generality, let $f$ be a model that receives a set $X$ of input data and provide predictions of the target response $y$. Input data can be monitoring data or pre-processed data after transforming monitoring data into features, whereas the response can be either a numerical value for regression, or a class for classification. A model is defined by a set $f$ that contains, among others, the type of algorithm and/or technique that characterizes the model and the needed parameters, e.g., ANN and all the parameters and coefficients of the trained model. In addition, the *meta-data* is coupled with the predictor and provides the context required to use properly the model. An example of meta-data is the characterization of the input features space

region, i.e., the range of each feature in the training data set. Then, before doing a prediction, those ranges should be checked to know if the input data is within the ranges observed during the training phase or, on the contrary, the model will potentially extrapolate the response.

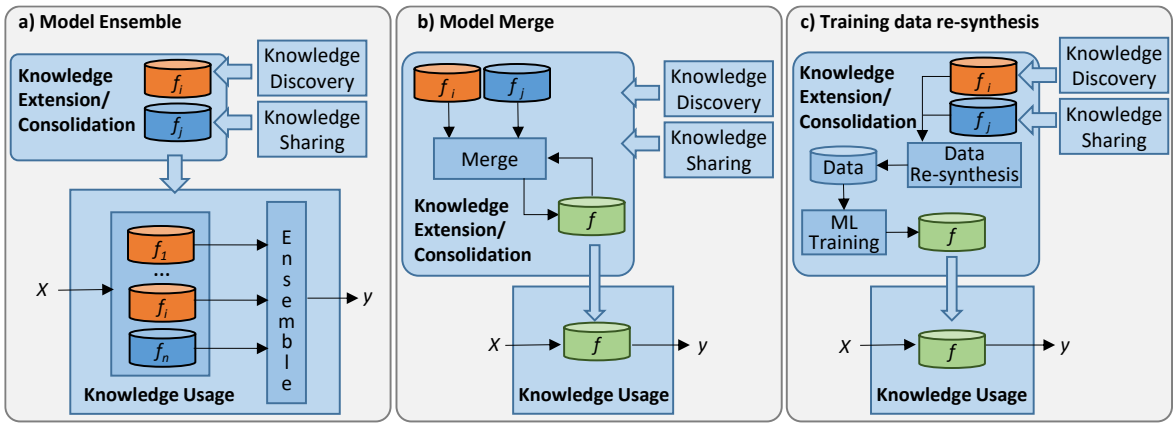


Fig. 4-4. Knowledge assimilation options: model ensemble (a), model merge (b), and training data re-synthesis (c).

### 4.2.1 Model ensemble

This method considers no just one single ML model, but a set (ensemble) of models for a problem that e.g., correspond to different feasible scenarios that can be observed. Thus, under a specific scenario, some models will produce accurate predictions, whereas some other will produce inaccurate ones.

Under the model ensemble method, when a new model is trained, e.g., for a new scenario, it is added into the set of models used by the problem (Fig. 4-4a). The new model will be used according to the output algorithm to generate one single output from the predictions made by a (sub)set of individual models in the ensemble. Under this option, the algorithm is the responsible of discerning how to combine and/or select individual predictions.

The combination of individual predictions can be done according to strategies as simple as using a weighted average of the individual responses according to some meta-data parameters that serve as weights. However, the availability of monitoring data enabling the dynamic evaluation of the individual predictions allow the implementation of adaptive voting procedures that can approach predictions to actual measurements [Di00]. Model ensemble is an option for knowledge extension that requires low computational effort and that can be applied to any ML technique and even combine different types of ML models. A mathematical description for both classification and regression applications is provided next.

Let $E=<f_1, f_2,\ldots,f_n>$ be the ensemble containing all available models for a given problem. Given an input data sample $X=<x_1, x_2,\ldots, x_m>$, we define the subset of

models $E^*(X) \subseteq E$ containing all the models within the region of the features space that contains $X$ that are eligible for predicting the response of the sample. This eligibility can be computed in terms of the probability that the sample belongs to the statistical distribution of that training data used to fit the model. Then, assuming that $\pi_i$ contains the characterization of the probability distribution of the input data variables of model $f_i$, such model can be included in $E^*$ if and only if $P(X \mid \pi_i) > \varepsilon$, where $\varepsilon \in [0,1]$ needs to be selected beforehand. A typical conservative configuration skipping those models whose training data statistical characteristics largely differ from sample $X$ could be $\varepsilon = 0.05$ [Re97].

Once the ensemble subset selection has been carried out, the individual predictions $y'$ are obtained for each model in $E^*(X)$, which are afterwards combined to produce a single combined prediction $y^*$. This combination is the result of applying a function that considers a weight $w_i \in \mathbb{R}^+$ for the prediction of every individual model $f_i \in E^*(X)$. In the case of classification where the response is one of the classes $c \in C$, $y^*$ is the class of the most common response considering the weights of the models. Specifically, $y^*$ can be computed as:

$$y^*(X) = argmax_{c \in C} \left\{ \sum_{f_i \in E^*(X)} w_i . (y_i == c) \right\} \tag{4-1}$$

In the case of regression, weighted average of the individual responses can be used, where $W$ is the sum of the individual weights:

$$y^*(X) = \frac{1}{W} . \sum_{f_i \in E^*(X)} w_i . y_i \tag{4-2}$$

Let us now focus on how the accuracy of the models can be evaluated. Let us assume that both individual and combined predictions are stored in the data repository (see Fig. 4-3) until the measured $y$ is available. Then, by comparing the measured $y$ with the individual predictions, the accuracy of each model in $E^*(X)$ can be evaluated. In particular, we define the subsets $E^*_{acc}(X)$ and $E^*_{ina}(X)$ as the accurate and inaccurate model subsets, respectively. Subset $E^*_{acc}(X)$ contains the models that produced good predictions, i.e., either those models that predicted the right class in a classification use case or those models that predicted a response within a confidence interval, e.g., 95%, in a regression use case. Note that $E^*_{ina}(X) = E^*(X) \setminus E^*_{acc}(X)$.

By classifying the models into accurate and inaccurate, we can dynamically update the individual weights $w_i$ used for combination purposes; minimum ($w_{min}$) and maximum ($w_{max}$) values are used to keep weights within a given range. Thus, the weight of inaccurate models can be reduced according to parameter $\rho \in [0,1]$, as:

$$w_i = \max(\rho . w_i, w_{\min}), \qquad \forall f_i \in E^*_{ina}(X) \tag{4-3}$$

whereas accurate models can be promoted by increasing its weight according to parameter $\tau \geq 1$, as:

$$w_i = \min(\tau . w_i, w_{\max}), \qquad \forall f_i \in E^*_{acc}(X) \tag{4-4}$$

Note that magnitudes and the cross-relation of $\rho$ and $\tau$ allow configuring different strategies, ranging from a long-term persistence of past accurate models to a short-term memory configuration leading to fast changes towards current good models.

### 4.2.2 Model merge

This method consists in merging individual ML models obtaining one single model for using the knowledge, which simplifies its operation (Fig. 4-4b). Note that the combination of model parameters in this method is key to assimilate the individual knowledge. Parameters of the joint model can be modified by the merging procedure as soon as new models are available. This methodology can provide potential benefits for those cases where model parameters can be partially updated without affecting the robustness and accuracy of the non-updated part.

For simplicity, in this section we focus on merging a pair of individual models based on linear SVMs in the context of a binary classification problem, where two classes are linearly separable; merging $n$ models can be defined as a concatenation of $n$-1 merge operations of model pairs.

Assuming that trained models $f_i$ and $f_j$ are linear SVMs, the coefficients of the decision hyperplanes of each model that perfectly divides the feature space region into two separated response classes can be easily obtained from the set of support vectors $V_i$ and $V_j$ [Sc01]. Then, let $B_i = [\beta^0{}_i, \beta^1{}_i, \ldots, \beta^m{}_i]$ and $B_j=[\beta^0{}_j, \beta^1{}_j, \ldots, \beta^m{}_j]$ be the vector of linear coefficients (i.e., the coefficient of every feature plus the intercept) of $f_i$ and $f_j$, respectively. Furthermore, in addition to meta-data $\pi_i$ and $\pi_j$ containing the statistical distributions of input features, the training data set size of every model (denoted as $s_i$ and $s_j$) is available.

The combined model, defined by the coefficients vector $B^*$, can be computed using equation (4-5), where the coefficients of the combined model are the weighted average of the coefficients of the individual models. Here, weights are computed by means of function $g(s)$ that depends on the number of training data samples of each model. Without loss of generality, we can assume that $g(s)$ is a simple transfer function such as the identity or the logarithm.

$$B^* = \left[ \frac{\beta_i^k \cdot g(s_i) + \beta_j^k \cdot g(s_j)}{g(s_i) + g(s_j)}, \qquad \forall k = 0..m \right] \tag{4-5}$$

Equation (4-5) produces a combined model regardless of the characteristics of the individual models. However, it is worth noting that models with dissimilar characteristics can produce inaccurate combined models. A simple but efficient procedure to avoid worsening the overall accuracy is to guarantee that the combined model stays within the margin hyperplanes of both individual models. Fig. 4-5 illustrates the proposed procedure for a simple example with just two input features. Fig. 4-5a-b show two initial models to be combined, where the decision and margin hyperplanes are depicted with solid and dashed lines, respectively. Hyperplanes are

depicted only in the range of the features observed for each variable; shadowed area in feature $x_1$ axis summarizes such range. In addition, the support vectors are depicted with markers on the corresponding margins, using a different marker shape for each class.

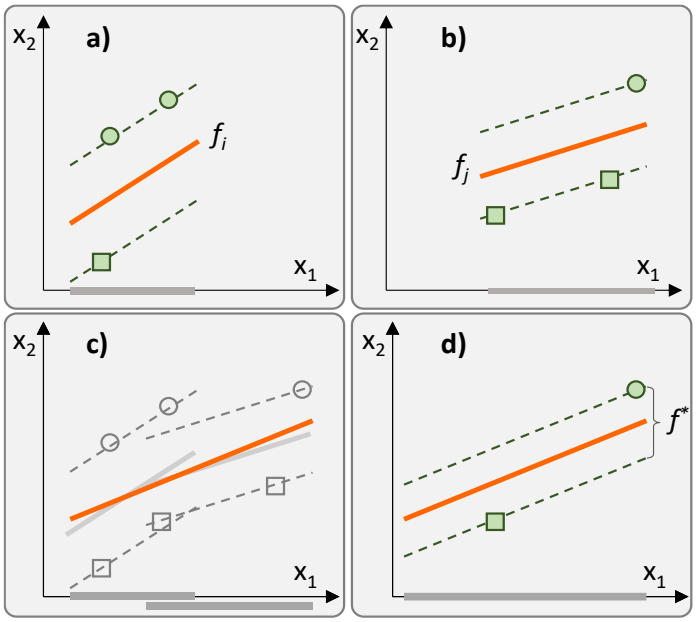


Fig. 4-5. Merging linear SVMs

By solving equation (4-5) and assuming $g(s_i)=g(s_j)$, the combined decision hyperplane is depicted in Fig. 4-5c, where the original margins and support vectors are depicted; we observe in Fig. 4-5c that the combined decision hyperplane remains within the margin hyperplanes of the individual models along the corresponding feature spaces and as such, the combined model does not lead to worse decisions. In consequence, to validate the combined model, one just need to verify that combined decision hyperplane and original margins do not intersect in the regions; otherwise, model merge cannot be performed with enough goodness-of-fit assurance.

Assuming that the merged model is validated, it is important to update the new margin hyperplanes and support vectors. To keep the main properties of SVM, margins can be generated by finding those parallel hyperplanes with respect to the decision hyperplane, such that intersect with the closest support vector/s. Fig. 4-5d shows the combined margins and the support vectors associated to the combined model.

Finally, recall that meta-data is required also for the combined model. Particularly, the region in the features space where such model can be applied is found by computing the union of the regions of the individual models.

### 4.2.3 Training data re-synthesis

Finally, this method consists in generating the response from the individual ML models in the given regions to obtain a synthetic training dataset from which a new ML model is trained (Fig. 4-4c). The training data re-synthesis from ML models enables reducing the amount of data being exchanged among agents, as well as the data being locally stored.

The synthetic data generation procedure needs to consider the specifics of both the problem and the techniques for modelling, to guarantee the persistence of the characteristics of the observed data. Note that some of the shared models and/or part of the synthetic data could need to be kept for future retraining cycles.

This option can be applied to both classification and regression problems. In the case of classification using SVMs, we need to guarantee that synthetic samples are not generated inside the space defined by margin hyperplanes. Indeed, data re-synthesis should be restricted to generating samples on the margins, i.e., synthetic support vectors. Fig. 4-6a illustrates an example where two linear SVMs cannot be merged due to the intersection of the combined decision hyperplane with one of the margins. When the re-synthesis method is applied, a number of synthetic samples on the margins of every model are firstly generated (transparent markers) to afterwards train a new SVM. Note that the SVM training algorithm finds the best SVM configuration, including the most proper kernel. This can be easily automatized by simply training with different kernels and returning the most accurate model. In Fig. 4-6a, a polynomial kernel has been chosen for the combined model in order to keep separable classes, where some of the synthetic samples generated become the support vectors of the combined model (solid markers).

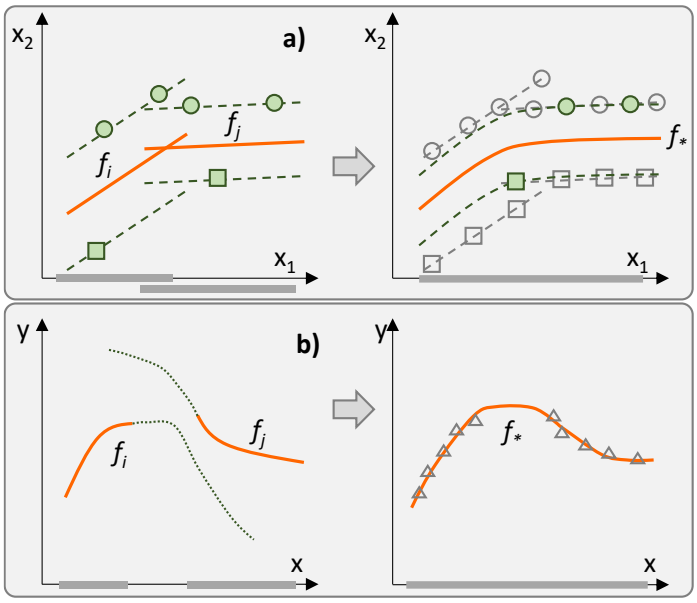


Fig. 4-6. Re-synthesis for classification (a) and regression (b).

In the case of regression, the synthesis of data points is performed by generating random samples that fit the statistical properties of the input region of the features space of every original model, e.g., following a Montecarlo approach [Kr11]. Then, the corresponding models are used to generate the response to label the sample. Once a significant amount of data samples has been generated for every model, the combined model is trained. Note that although in this chapter we use ANN for regression, the above procedure can be applied to other techniques.

Fig. 4-6b shows a simplified regression problem where one single feature is used to predict the response *y*; two non-overlapping models are to be combined. Dashed lines illustrate how inaccurate each model can be when it is used for prediction using as input a data point that it is outside its region of the feature space (*extrapolation*). On the contrary, the combined model once trained from synthetically generated data samples (depicted as triangles) preserves the goodness-of-fit of both individual models.

As a conclusion, every method described in this section for knowledge assimilation has its pros and cons, which makes that the method fits better in some use cases than in others. Table 4-1 summarizes the main pros and cons of extension and consolidation methods.

*Table 4-1. Pros and cons of knowledge assimilation methods*

| | |
|---|---|
| **Extension** | |
| • Model Ensemble | **Pros**: negligible assimilation complexity<br>**Cons:** High storage and complex knowledge usage |
| **Consolidation** | |
| • Model Merge | **Pros:** Low storage and simple knowledge usage<br>**Cons:** High assimilation complexity (algorithmic) and risk to degrade model accuracy |
| • Data re-synthesis | **Pros:** Simple knowledge usage<br>**Cons:** High storage and high assimilation complexity (computational). |

## 4.3 Use Cases

In view of Table 4-1, this section defines two borderline use cases for illustrative purposes, where the architecture for KM and the methods for knowledge sharing and assimilation presented in the previous sections are applied. The first use case uses KM in a *purely distributed* scenario, where knowledge is shared among the different network nodes, whereas the second use case uses KM in a *purely centralized* scenario, where although knowledge is shared among models, the whole KM process is entirely carried out in the MDA controller running besides the SDN controller. The use cases highlight the flexibility of the proposed architecture for KM, which can

be easily adapted for different applications in multilayer network scenarios. In fact, the placement of knowledge components has been forced to fit these two borderline use cases, but it does not preclude other configurations to be feasible and even better in terms of performance. These use cases will be considered in the next section for the validation of the proposed architecture.

The architecture of the purely distributed use case is represented in Fig. 4-7a and is based on the autonomic transmission application in [Ru19], where an autonomic agent running in the optical transponders collects and processes SOP and pre-FEC BER monitoring data at a rate of one sample every 278 μs, and it is able to anticipate QoT degradation caused by fiber stressing events. The prediction anticipates such degradation tens of ms before it actually happens by applying properly trained ML models; the output is used to configure the number of iterations to be performed by the error correction algorithm in the FEC module.

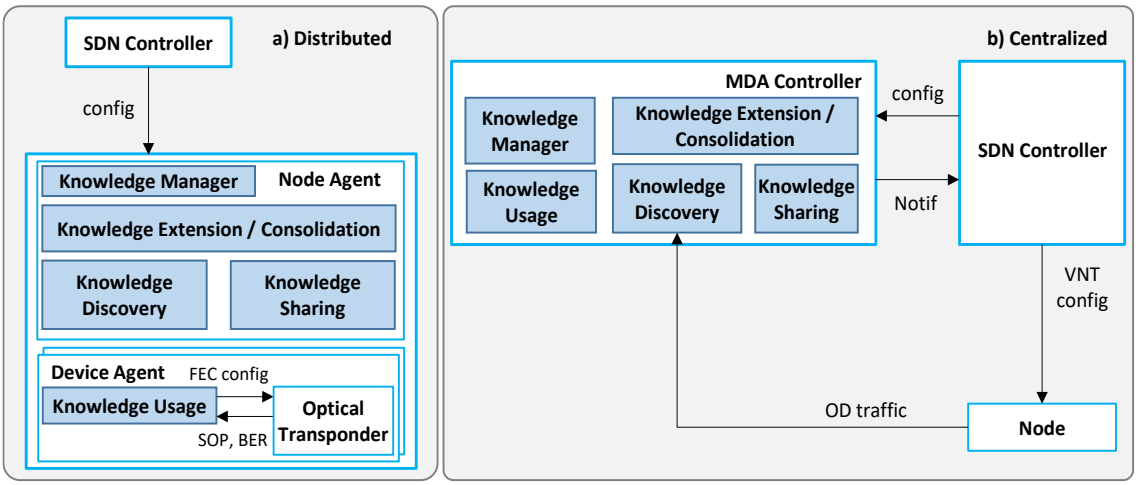


Fig. 4-7. KM applied to the purely distributed (a) and centralized (b) use cases.

In this use case, it is clear the need of adopting continuous learning, justified by the impossibility to accurately train ML models to predict every possible physical fluctuation for all possible network scenarios before entering into operation. Moreover, since similar SOP fluctuations are plausible to happen in different links at different time, the benefits of sharing knowledge are expected to be high, as the relationship between SOP fluctuations and QoT in the event of gusts of wind in aerial fiber cables can be learnt in some part of the network and shared among the nodes.

In addition, knowledge usage needs to be embedded into the device agent due to the extremely high data collection rate and the need of rapid decision making and device configuration; it is a case of device-level control loop. Regarding knowledge discovery, recall that it entails analyzing predictions and real measurements to find inaccuracies (wrong classification) that could lead to training new ML models. The placement of this component cannot be done neither in the device agents because of their limited computational resources, not in the centralized SDN controller because

of the large amount of data to be transferred. In this case, the node agent seems the most proper place to deploy the knowledge discovery component. Consequently, knowledge sharing is carried out among the node agents that exchange models and/or data and implement knowledge assimilation to complete the KM process.

The architecture of the purely centralized use case is represented in Fig. 4-7b and is based on the autonomic VNT reconfiguration in [Mo17.2]. OD traffic monitoring samples are collected from the packet nodes in the network and used to predict the OD traffic expected for the next time interval, e.g. 1 hour. Traffic prediction is used to feed a VNT re-optimization problem that finds the best VNT configuration for the forecasted OD traffic matrix [Ve17.1].

Here, a variety of reasons, like the continuous traffic increment, the introduction of new services with strong requirements, etc., make KM process implementation for continuous learning to be a good choice. In this use case, although different architectures could be feasible, the network-wide control loop entails that knowledge components are located in the MDA controller. Hence, monitoring traffic data can be collected at a coarse interval, e.g. 15 minutes, and analyzed in the MDA controller for dynamic VNT reconfiguration purposes. Continuous learning is needed to adapt models to traffic evolution; here an inaccuracy is defined as a prediction with error above some defined threshold. Notwithstanding the centralized architecture, knowledge sharing can be carried out among OD traffic models; here knowledge assimilation based on data exchange can be an option, in the case of enough storage is available. The selection of the subset of OD to whom share knowledge is also important in the case of ODs can be classified as a function of the type of traffic they convey.

Finally, note that in both use cases, the SDN controller should be in charge of setting the proper configuration parameters and policies for the KM process. In particular, policies should specify *what*, *when*, *how*, and to *whom* knowledge needs to be shared, *when* knowledge assimilation should be carried out, etc.

## 4.4 Results

In this section, we first introduce the simulation environment used for performance evaluation and define the specifics of the two selected use cases. Next, we study and compare the performance from applying KM and start by considering KM based on data exchange, where data related to the detected inaccuracies is distributed, as well as based on model exchange, where the knowledge assimilation techniques presented in Section 4.2 are applied.

### 4.4.1 Simulation Environment and Use Cases

For performance evaluation of the proposed KM process, a simulation environment has been developed in R. A network consisting of a number of nodes, each composed of several devices, and connected by a set of links is reproduced. Specifically, we configured a scenario reproducing a small-size metro network consisting in 10 locations, where each location consists of both a packet node and an optical node each equipped with 10 ports.

Initial datasets for each use case were generated based on the topology characteristics and end-users information from [Metro-Haul18] and initial ML models for each device were trained. Each device includes a data generator to synthetize monitoring data for the target use case. Operation was emulated by generating synthetic monitoring samples that include events that were not observed during the initial ML training phase, so new knowledge is discovered.

In the case of the purely distributed autonomic transmission use case, devices emulate optical receivers and generate synthetic monitoring samples at a rate of 278 μs (3600 samples/s). Each sample consists of a 42-byte tuple <*t*, *S*, *BER*>, where *t* is the timestamp, *S* is the set of values of the three Stokes parameters, and *BER* is the pre-FEC BER measurement. Realistic fiber stressing events causing correlated SOP and pre-FEC BER fluctuations were randomly generated based on the experimental measurements carried out in [Ru19]. For this use case, we considered SVMs to predict the proper configuration of the FEC module (i.e., number of FEC iterations) as a function of pre-computed features gathering the current value and trend of each of the Stoke parameters. Note that those features can be easily pre-computed from the generated synthetic monitoring data [Ru19]. Finally, an inaccuracy is defined as a misclassification, i.e., the model predicts a wrong number of FEC iterations.

For the purely centralized autonomic VNT reconfiguration use case, devices emulate network interfaces in packet nodes. We used the CURSA-SQ methodology (see Section 2.3) to generate realistic packet traffic flow samples with granularity 15 minutes, emulating the monitoring data collected from those interfaces. Each sample consists of a 64-byte tuple <*t*, *OD*, *B*>, where *t* is the timestamp, *OD* is a string identifying the OD flow, and *B* is the bitrate measurement in b/s. OD traffic is predicted using ANNs whose inputs are the measurements in the last hour and the number of hidden neurons equals to the number of inputs, in line with the modelling approach presented in [Mo17.2]. Here, an inaccuracy is defined as a prediction for which the magnitude of the error for a real measurement is greater than the percentile 95% of the error observed during the training phase.

The simulation environment follows the KM architecture proposed in Fig. 4-3, where the different KM components can be placed in node agents and/or the MDA controller to compose the distributed and centralized scenarios presented in Fig. 4-7, as well as any other intermediate configuration. Moreover, the configuration of the policies for knowledge discovery, assimilation, and sharing can be configured from the SDN

controller. Finally, the MDA controller collects relevant network performance evaluation data, including the evolution of the accuracy of the models and the amount of shared data.

### 4.4.2 Data-based Knowledge Management

Let us first evaluate the performance of KM based on sharing data. We assume that inaccuracies are shared when they are detected. Specifically, we consider two different policies for data sharing: *i*) *inaccuracies*, where inaccurate data points are shared and *ii*) *extended data*, where inaccurate data points go hand in hand with other data points. In *inaccuracies* policy specifically, we consider that a small window of samples (e.g., 30 samples) is needed to be shared to compute the features for the inaccuracy in the case of the purely distributed autonomic transmission, whereas just one sample is needed in the case of the purely centralized VNT reconfiguration. Note that this policy is adapted from the collective self-learning approach presented in [Ve19.1]. In *extended data*, those points that were not been identified as inaccuracies, could be potentially useful to improve ML models. Although other options could be considered for selecting such additional data points, an extended window to allow compute the evolution of the features is shared in the case of autonomic transmission, whereas individual samples measured immediately before the inaccuracy are shared in the case of VNT reconfiguration. The amount of additional data points that provides the best trade-off between accuracy and data volume depends of the use case and scenario and it will be analyzed. Finally, ML model re-training is carried out periodically, e.g., every hour, provided that inaccurate data points are available.

Fig. 4-8 shows the performance of the proposed data-based KM in terms of the evolution of the prediction error against emulated operation time, for both the purely distributed autonomic transmission (Fig. 4-8a) and the purely centralized VNT reconfiguration (Fig. 4-8b) use cases. For benchmarking purposes, we included the performance of no sharing knowledge. For convenience, prediction error has been normalized to the error of the initial models, whereas operation time was normalized to the time when the least accurate approach reaches a low target error (e.g., 0.1%). Interestingly, the results show similar behavior for both use cases, where large benefits from knowledge sharing are observed. In particular, extended data sharing shows a better convergence time, reaching the target error 60-70% faster than without sharing knowledge. Moreover, is that policy the only one that achieves negligible errors around 0.01%. The inaccuracies sharing policy shows also excellent performance and although its convergence time is above than that of the extended data sharing one, it is over 35% faster than no sharing knowledge. In fact, both data sharing policies show a similar error evolution until reaching error around 3-4%, which makes that the policy selection needs to be based on other criterion in case the target error criterion can be relaxed.

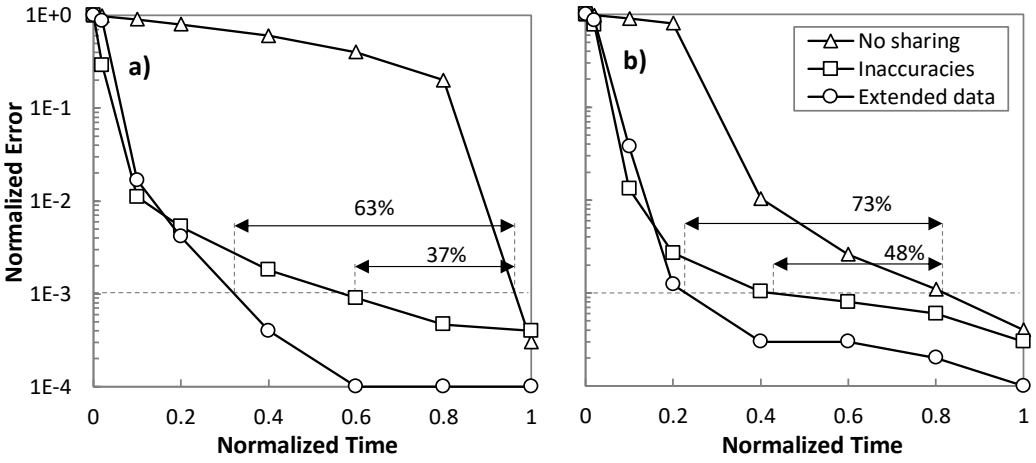


Fig. 4-8. Data-based KM performance for the distributed (a) and centralized (b) use cases.

In fact, particular interest should be payed to the amount of total data that is shared. This criterion is relevant mainly for the purely distributed use case, as such data is exchanged among agents that are not in the same location. Table 4-2 shows the gain in terms of convergence time as a function of the data shared per inaccuracy for the purely distributed and the purely centralized use case. The multiplier refers to the amount of additional data that is shared, where x1 is equivalent to the inaccuracies sharing policy. The amount of additional data that needs to be exchanged to achieve the gains showed in Fig. 4-8 represents an increment of 3 times (x4) the amount of data exchanged with the inaccuracies sharing policy.

*Table 4-2: Convergence Time Gain w.r.t No Sharing (%)*

| | **Multiplier of shared data per inaccuracy** | | | |
|---|---|---|---|---|
| | **x1** | **x2** | **x3** | **x4** |
| **Distributed** | 36.8 | 45.5 | 61.25 | 63.2 |
| **Centralized** | 47.5 | 66.25 | 71.3 | 72.5 |

Fig. 4-9a and Fig. 4-9b show the number of inaccuracies and the total data volume shared during the entire simulation as a function of the amount of data exchanged per inaccuracy for the purely distributed and the purely centralized use case, respectively. The evolution of the total number of inaccuracies shows how they are reduced when the amount of extended data is increased (about 1/3 in the case of the distributed and 60% in the case of the centralized use case). Such reduction is the base of the achieved convergence gain. Regarding the amount of total data shared, although acceptable for the purely centralized use case, it is above 1 GB for the purely distributed one. Recall that every accuracy entails 30*42 bytes in the case of autonomic transmission, and 64 bytes in the case of VNT reconfiguration to be

shared with ($10^{10}$) agents. Even with the reduction of the number of inaccuracies, the volume of exchanged data is high under the extended data policy.

In view of these results, and considering that the probability of discovering inaccuracies decreases with time, a mixed data-based approach can be followed; the inaccuracy sharing policy can be first applied to allow an initial fast convergence with a reasonable amount of data being shared, followed by the extended data sharing policy after reaching a certain error level to increase even more models' accuracy.

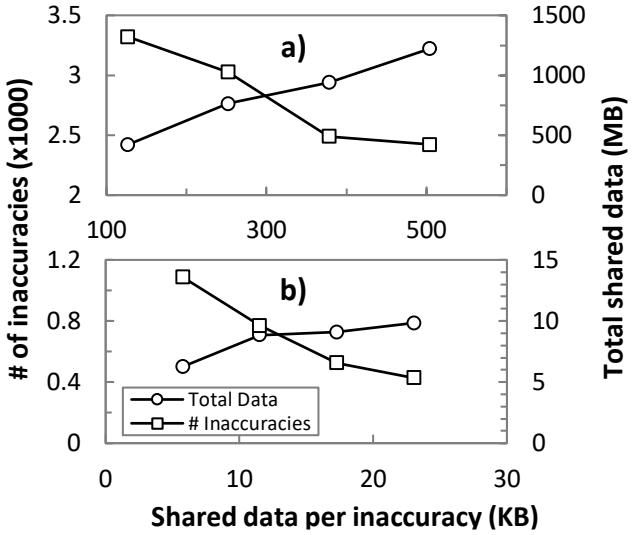


Fig. 4-9. Extended data policy analysis

### 4.4.3 Model-based Knowledge Management

Let us now explore policies based on sharing models and knowledge assimilation by means of the methods proposed in Section 4.2. Recall that, in addition to the models, meta-data is needed to specify their region of applicability; specifically, we limit meta-data to specify the range (minimum and maximum) of each input feature. For the ongoing analysis, we assume that the model ensemble method is configured with a short-term memory tuning. Specifically, the following configuration was chosen: $\rho$=0.6, $\tau$=1.5, $w_{min}$=1, $w_{max}$ =10. Regarding model merge and training data re-synthesis, we used them according to the characteristics of the ML techniques used for the purely distributed (SVM) and purely centralized (ANN) use cases, respectively.

Aiming at evaluating the performance of different policies and the impact of the main blocks involved in knowledge assimilation, i.e., knowledge extension and consolidation, we compare three basic policies: *i*) *extension*, where every new shared model is added to a device models pool and used together with the model ensemble method, without any consolidation action; *ii*) *consolidation*, where just a single model is maintained, i.e., incoming shared models update the model by either model merge

or training data re-synthesis methods, depending on the use case; *iii*) *extension and consolidation*, where both knowledge extension and consolidation is continuously performed to keep moderated the size of the models' pool (we limited its size to 10 models). Meta-data is used to join models within a region or, if necessary, in nearby regions of the features space.

Fig. 4-10 shows the evolution of model error against time for the above model-based policies and the defined use cases. For the sake of comparison, we included the two data-based KM policies previously analyzed in Fig. 4-8. The results show that the policy combining knowledge extension and consolidation achieves a performance comparable to that of the data-based KM thus, validating in terms of accuracy a KM process based on sharing models instead of monitoring and pre-processed data. The other two policies show worse performance and lead to either an increasing number of models, which makes difficult a practical operation, because of the large number of models, and reduces the potential of incremental learning, or to a forced consolidation, which combines models with dissimilar characteristics in different regions, which increases errors that reduce the gain obtained by the acquired new knowledge (see Section 4.2.2).

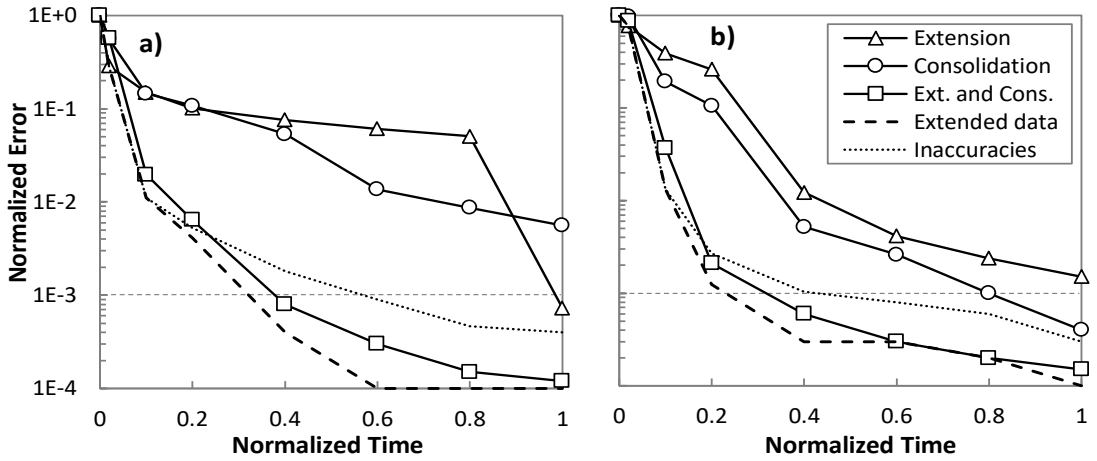


Fig. 4-10. Model-based KM performance for the distributed (a) and centralized (b) use cases.

Once the excellent performance of the model-based KM with the policy combining knowledge extension and consolidation has been demonstrated, its practical applicability depends mainly on the amount of data involved in knowledge sharing, as compared to data-based KM policies. Fig. 4-11a and Fig. 4-11b show the evolution of the ratio between the data shared by each data-based policy and the combined policy of the model-based one as a function of model errors for the distributed and centralized use cases, respectively. Ratio equal to 1 (highlighted as a dashed line) represents the case where data-based and model-based policies exchange the same amount of data, whereas when the ratio is lower than (higher than) one entails data-based (model-based) policy exchanging less data. As it can be observed, data-based

policies provide benefits in terms of exchanged data only when very low error are achieved.

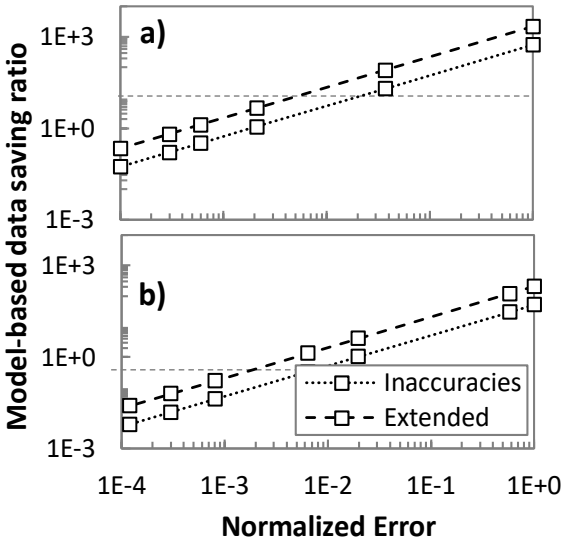


Fig. 4-11. Data sharing comparison

In the rest of cases, model-based KM reduces the amount of shared data several orders of magnitude in both the distributed and centralized use cases. Table 4-3 complements Fig. 4-11 and presents the total amount of data exchanged at the end of simulations by each of the policies for each of the use cases.

*Table 4-3: Total amount of shared data (in MB)*

| Use case | Model-based | Min data reduction | Data-based (inaccuracies) | Data-based (extended) |
|---|---|---|---|---|
| **Distributed** | 2.6 | 99% | 333.4 | 1333.5 |
| **Centralized** | 1.5 | 90% | 15.2 | 60.9 |

As a conclusion, the combined knowledge extension and consolidation policy of model-based KM provides virtually the best performance and it is the most scalable option by far. Nevertheless, one can combine different policies by selecting the one that better fits the current scenario. In particular, the selection of data-based and model-based policies at different times of the KM process as a function of model's accuracy could provide the best performance. This is highlighted in Fig. 4-12a and Fig. 4-12b, where a mixed strategy combining data-based and model-based policies are compared in terms of the total amount of shared data for the distributed and centralized use case, respectively. According to the performance results in Fig. 4-10, the mixed policy providing the optimal performance would consists of the model-based policy for the initial phase until models reach error around 1%, followed by the data-based inaccuracies policy, until the error reaches around 0.1% and

complemented by the extended data-sharing policy to reach a negligible error around 0.01%. As it can be observed, the mixed policy allows reducing even more data volumes involved during knowledge sharing.

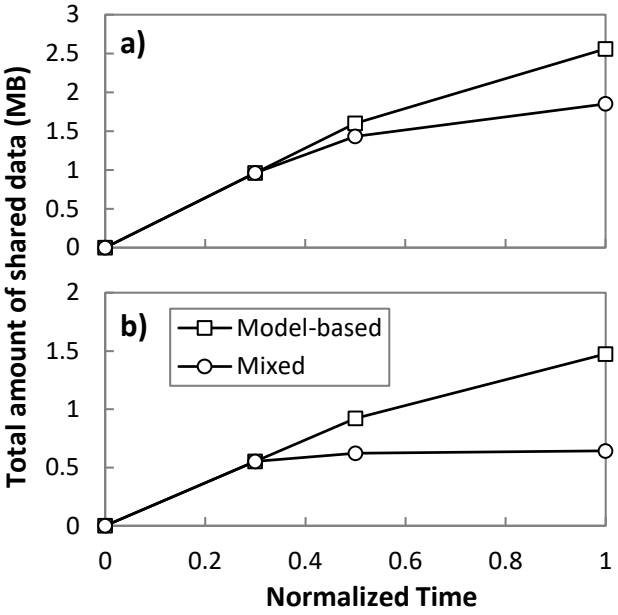


Fig. 4-12. Model-based and Mixed knowledge sharing

## 4.5 Concluding Remarks

The KM process has been proposed aiming at a truly autonomous optical network operation. KM is based on four main pillars: *i*) knowledge discover; *ii*) knowledge share; *iii*) knowledge assimilate; and *iv*) knowledge usage. These pillars allow optical networks to autonomously discover and disseminate knowledge that can be used to adapt its configuration to variable conditions without human intervention.

A general architecture to support KM has been proposed that extend beyond typical control loop implementation and allows for knowledge sharing among different agents disregarding they run distributed in the network nodes or centralized in a controller, like the Monitoring and Data Analytics (MDA) one. Such knowledge sharing enables collective self-learning, which has been demonstrated to reduce models error convergence time.

However, knowledge sharing entails data distribution and storage and hence, keeping limited the amount of data is a key issue. In that regard, two alternative strategies consisting on the distribution of data samples related to model inaccuracies (data-based) and models representing such inaccuracies (model-based) are studied. For the latter strategy, three methods for knowledge assimilation are proposed: *i*) model ensemble, *ii*) model merge, and *iii*) training data re-synthesis.

With these methods, knowledge assimilation can be implemented by means of two main actions to manage ML models: extension and consolidation. Such actions are carried out in the knowledge assimilation component in the architecture. In particular model ensemble, allows an efficient and accurate use of ML model pools, model merge allows combining the coefficients of different models to produce a combined model and training data re-synthesis allows to consolidate different models based on regenerating data from them that are used to train new models.

Two illustrative use cases have been used to illustrate the potential application of the KM architecture and to evaluate different policies for knowledge sharing and assimilation: *i*) the purely distributed autonomic transmission use case, where knowledge is used at the optical transponder system level and knowledge sharing and assimilation is carried out at the node level; and *ii*) the purely centralized VNT reconfiguration use case, where all the components run at the MDA controller level. Note that even in this case, a different model is kept for every of the origin-destination traffic flows in the VNT, so knowledge sharing and assimilation takes also place.

The KM process has been evaluated by simulation on a metro network scenario for the defined use cases in terms of model error convergence time and amount of data shared among agents. Two different data-based policies were studied and concluded that sharing data inaccuracies and retraining ML models leads to a fast error convergence time until reaching a certain low error, where error can be reduced even more when additional (extended) data was shared along with the inaccuracies. In addition, a model-based policy based on applying coordinated extension and consolidation actions demonstrated similar convergence time than data-based policies with few orders of magnitude less of data being shared among agents. Indeed, the combination of the three data-based and model-based policies at different phases of the network learning process reached minimal shared data volumes without compromising the convergence towards highly accurate models.

The next chapters elaborate and extend the KM concepts just developed on different network scenarios.

# Chapter 5

# Modeling and Assessing Connectivity Services Performance

The automation of NS consisting of virtual functions connected through a multilayer packet-over-optical network requires predictable QoS performance, measured in terms of throughput and latency, to allow making proactive decisions. QoS is typically guaranteed by overprovisioning capacity dedicated to the NS, which increases costs for customers and network operators, especially when the traffic generated by the users and/or the virtual functions highly varies over the time.

This chapter presents the PILOT methodology for modeling the performance of connectivity services during commissioning testing in terms of throughput and latency. Benefits are double: first, an accurate per-connection model allows operators to better operate their networks and reduce the need for overprovisioning; and second, customers can tune their applications to the performance characteristics of the connectivity.

PILOT runs in a sandbox domain and constructs a scenario where an efficient traffic flow simulation environment, based on the CURSA-SQ model (see Section 2.3), is used to generate large amounts of data for ML model training and validation. The simulation scenario is tuned using real measurements of the connection (including throughput and latency) obtained from a set of active probes in the operator network. PILOT has been experimentally validated on a distributed testbed connecting UPC and Telefónica premises.

## 5.1 Modeling and Assessing Connection KPIs

In this section, we detail our proposal for modeling and assessing connection performance, based on one-way active network measurements. Fig. 5-1 presents an infrastructure based on the one described in Section 2.1, where a multilayer network interconnects three COs. In the COM plane, an NFVO deals with the deployment of NFV NSs. The multilayer interconnection network is controlled by a hierarchical SDN architecture, where a Parent controller is on top of per-layer SDN controllers, one for the packet and another for the optical layer. Finally, an MDA controller collects monitoring data from the underlying infrastructure, trains ML models in the sandbox domain, and applies data analytics techniques on these data.

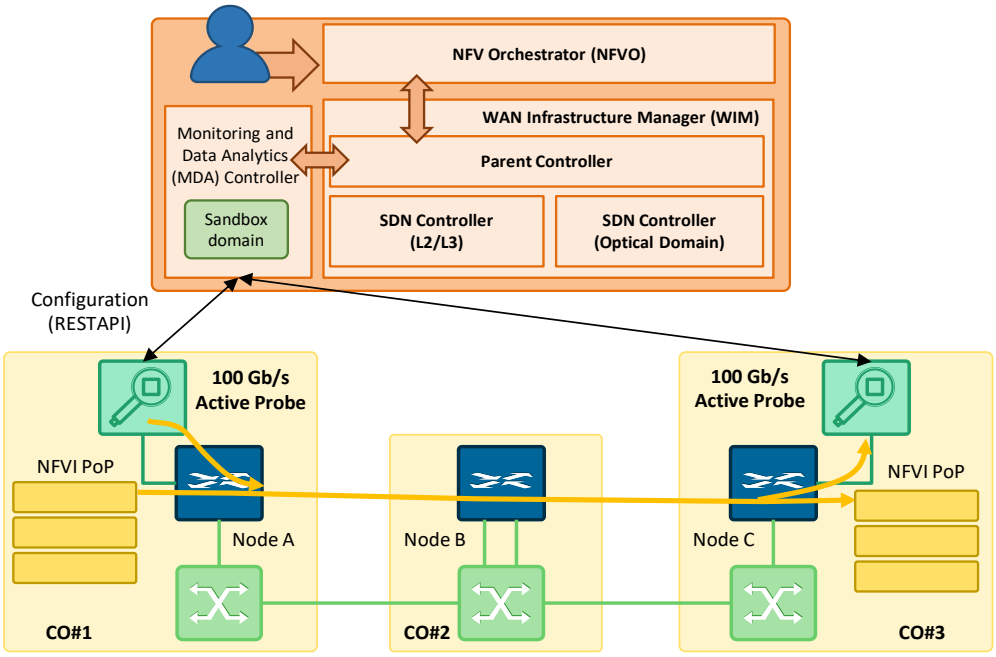


Fig. 5-1. Reference architecture for Active Monitoring.

Given that one of the main aspects of network automation is to guarantee that provisioned connections actually meet the requested performance, active probes are equipped at the packet layer to measure the performance of packet connections. The active probes are installed in every CO and connected through a 100GbE interface of a L2 switch in the internal CO network. Being the active probes connected to interfaces configured in trunk mode, the probes can tag the generated Ethernet frames with the desired VLAN ID, selecting the VLAN to be measured. The active probes have been developed to be integrated in the above described COM system. To that end, they expose a REST-API-based northbound interface (NBI) through which the MDA controller can configure them and initiate a measurement session on a specific packet connection.

In the example in Fig. 5-1, a simple NS that consists of two VNFs in CO#1 and CO#3 interconnected by a unidirectional packet connection is shown. The packet circuit has resources reserved in Nodes A, B and C, and its performance is measured by the active probe in CO#1 that acts as a sender and CO#3 that acts as a receiver. Although the active probes provide really accurate measurements between them, note that because they are connected to the packet node that provides the external connectivity to the CO (Node A and C), those measurements do not include parts of the network, like the internal PoPs' networks. For these very reasons, as well as to generate the large training and validation dataset, we propose to use CURSA-SQ to emulate the complete connectivity set-up and to generate useful models for the customers. However, real measurements are strictly needed to tune the CURSA-SQ scenario, which includes *additional* delays (e.g., transmission and other processing-related delays, see [Fin19] for a complete delay model) and fine tune of the queues.

Fig. 5-2 illustrates the above concepts, where the active probes measure the performance of the circuit between Node A and Node C. The proposed PILOT methodology runs inside a sandbox domain in the MDA controller, and it includes a module to configure the probes to perform the required measurements based on the characteristics of the services provided by the customer using two random variables (inter-arrival burst rate -IBR- and the burst size -BS) (step A in Fig. 5-2); the resulting measurements are collected and used to tune a network simulator based on CURSA-SQ (B). Once enough data is generated (C), ML models are trained and validated (D) and they can be shared to the connection's customer (E), which will use them to estimate performance metrics based on the load (F). Note that the produced ML models provide a way to reproduce the behavior of the connection without revealing the internal routing or other network details, which facilitates being shared to end customers. An alternative to ML models would be providing abstracted end-to-end performance data, at the cost of moving large volumes of data.

To illustrate the network simulation process, Fig. 5-3a shows an example of a slightly more complex p2mp connection defined between a source VNF and a set of destinations, i.e., <*src*, {*dest1*, *dest2*, *dest3*}>. Let us assume that the customer has specified that such p2mp connection will be used to convey a set of services *S* (in the example *S*={*s1*, *s2*, *s3*}). The details of the p2mp connection received from the Parent SDN controller include route of the tree, the resources actually reserved, the traffic specifications, and the QoS constraints. In the sandbox domain, the PILOT algorithm defines the scenario for CURSA-SQ-based simulation (Fig. 5-3b). The CURSA-SQ scenario includes a traffic generator (G) in the source VNF for each of the services specified, a sink node (Sk) for each VNF destination, dimensioned queues for each output interfaces in the route of the tree, and delay nodes (d) emulating the delay introduced by the links.

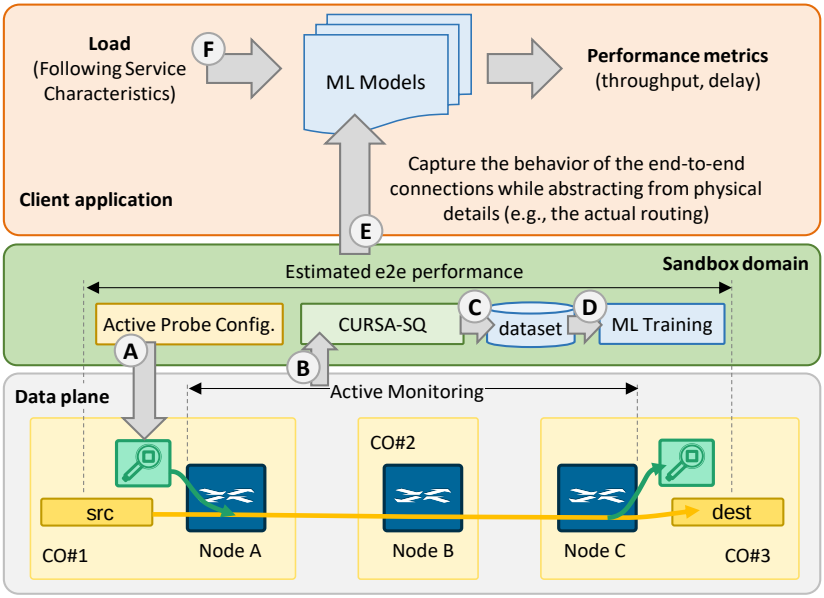


Fig. 5-2 Proposed Sandbox domain.

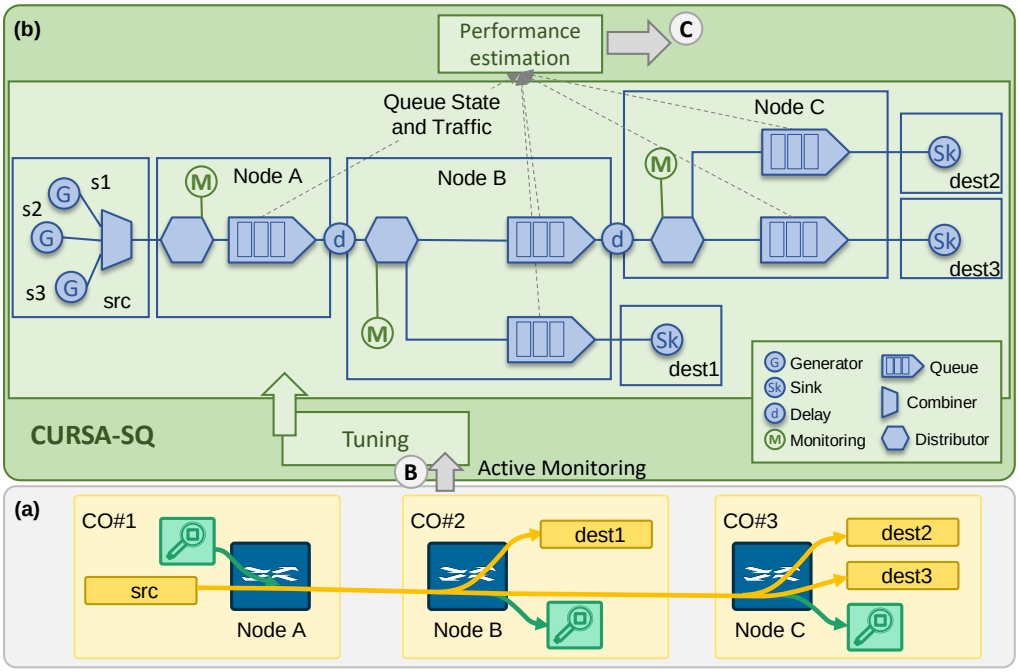


Fig. 5-3. Example of p2mp connection (a) and CURSA-SQ-based simulation (b).

Next, PILOT configures the active probes to measure the performance at every destination CO in the packet connection that will be used to tune the CURSA-SQ scenario, e.g., to ensure that any additional delay is included in the real set-up. Furthermore, in the case of p2mp connections, the probes join a multicast group created by the Parent SDN controller specifically for the commissioning tests; the source probe uses the multicast group as destination IP address for the generated

packets. Once the probes are configured, PILOT generates measurements configurations that include the definition of bursts mimicking the specified mix of services at meaningful values of IBR and BS random variables. Once the results are received from the destination probes (step B), PILOT uses them to tune the simulation scenario and runs CURSA-SQ to generate a large amount of labeled data for ML training and validation (C). The next section describes the PILOT methodology in depth.

## 5.2 Combining Measurements and Synthetic Data

The general scheme of the PILOT methodology is sketched in Fig. 5-4. PILOT entails three sequential stages to be carried out to produce accurate ML models for each packet connection. PILOT relies on the specification of the traffic mix that the connection will support. Specifically, the mix of traffic is defined in terms of services characterized by, at least, IBR and BS random variables, and a scaling factor. Such specification of the traffic mix is used to generate meaningful active probe configurations in terms of packet trains that are generated by the active probes and which measurements are used to tune the CURSA-SQ scenario. Once experimentally assessed, synthetic data that reproduces the real connection is generated, and accurate ML models can be trained and validated. The following sub-sections elaborate on key PILOT methodology components.

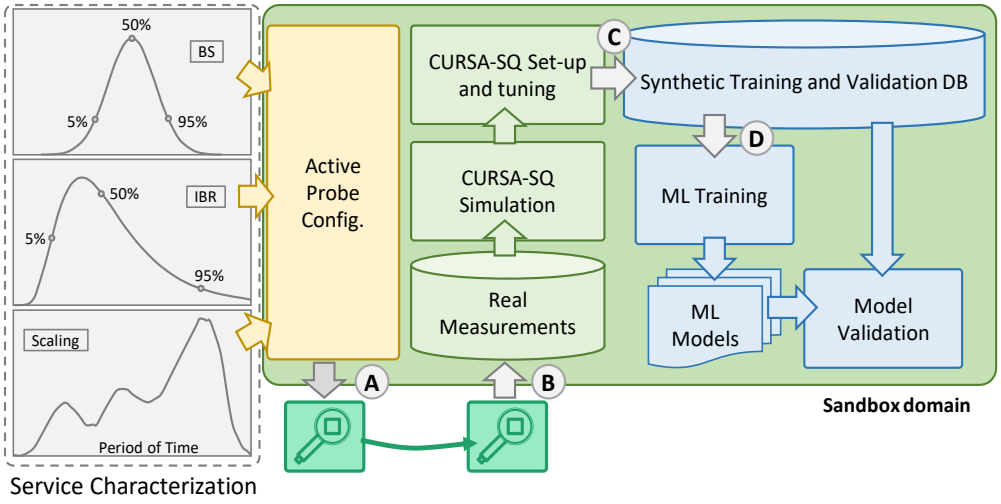


Fig. 5-4. Overview of the PILOT Methodology.

### 5.2.1 Traffic mix specification

As introduced in the previous section, we assume that a traffic specification is received for each connectivity request. Such specification includes the characterization of the set services $S$ that the connection will support. Service

characterization must include, at least, the statistical distribution and associated parameters of the IBR and BS *burst-level* random variables plus the scaling. The characterization can be provided by the customer or the network operator, and can be based on specific measurements or on studies available in the literature (see [Ru18]). From the received service characterization, we define the traffic specification $\chi_s$ for service $s$, as follows:

$$\chi_S = \{IBR_S \sim f(\theta_S), BS_S \sim g(\vartheta_S)\}, \forall s \in S \quad (5\text{-}1)$$

where $f$ and $g$ denote probability distribution functions with their respective parameters. In line with [Ru18], we consider that IBR and BS can be treated as independent variables; indeed, $f$ and $g$ can belong to distinct families of probability distributions. In addition, the characterization of the services at *packet-level* can lead to a more precise configuration of active probe measurements. In that regard, *packet size* (PS) is an additional random variable that could be included in $\chi_s$.

The expected demand needs also to be included for scaling of each service. We denote this input as $u_s(t)$, where the scaling (e.g., number of individual users) of each service $s$ is defined as a function of time. The specific time range is defined by the customers according to their interests, and it can cover from hours/days/weeks (e.g., a typical tidal profile that periodically repeats on time) to months (e.g., the expected user evolution during connection lifetime). Note that this flexible definition of the expected load opens the possibility to carry out several analysis leading to different KPI modelling for short, medium, and long-term applications. For the sake of simplicity, we assume the same time range for all the services in a connection.

The statistical properties of the services and their expected demand in time are key to understand and define the *traffic flow* $x(t)$ (bitrate, defined in b/s) injected into the connection. For modelling and simulation purposes (mainly for generation), we consider to model services separately, and therefore, the expectation ($E$) and variance ($V$) of each service in the flow can be computed as follows:

$$E\big(x_s(t)\big) = u_s(t) \cdot E(BS_S) \cdot E(IBR_S) \quad (5\text{-}2)$$

$$V\big(x_s(t)\big) = u_s(t) \cdot V(BS_S \cdot IBR_S) \quad (5\text{-}3)$$

where the variance of the product of BS and IBR can be estimated from well-known approximations of the variance of the product of two independent variables [Ca02]. Note that the connection traffic flow $x(t)$ is the aggregation of all $x_s(t)$.

### 5.2.2 Traffic Sampling and Measurements Configurations

Let us now detail the procedure for sampling the traffic flows $x_s(t)$ to obtain real measurements for those traffic samples using the active probes under realistic traffic conditions (see Fig. 5-4). We have redefined the synthetic packet generation in the active probes for this purpose, where a measurement request is defined by a number

of packet bursts, each containing packets of a given size. The definition of the bursts and the delay between two consecutive ones can be defined to reproduce a desired traffic pattern. The objective is then to define how measurement configurations are created to follow the main statistical characteristics of the specified traffic mix for the connection.

Fig. 5-5 summarizes the concept behind the generation of measurement configurations. Service characteristics are processed by a sample generator module that generates a set of samples of a given duration of bursty traffic; the duration, e.g., 5 ms, is defined by the capacity of the connection.

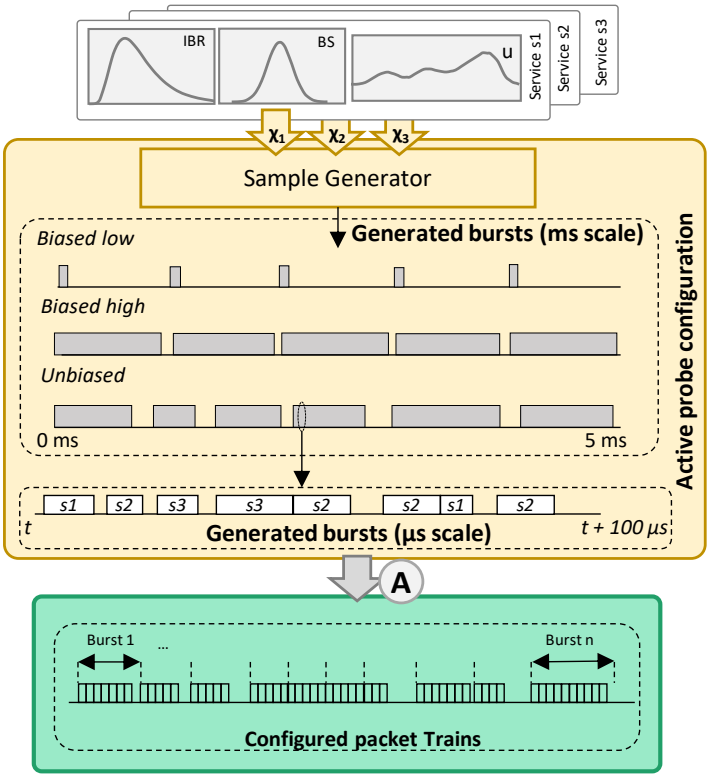


Fig. 5-5. Active probe configuration procedure.

The sample generator module firstly computes the expectation and variance of IBR and BS random variables based on their probability distributions. Then, several time values conveniently spaced in the time range of $u_s(t)$ are selected, thus covering relevant traffic mixes for low, medium, and high loads. For each selected time and mix, samples are generated according the expectation and variance references. In particular, three classes of samples are considered: *i*) *unbiased* samples, where $E$(IBR) and $E$(BS) are used for all the services; *ii*) *biased low*, where both $E$(IBR) and $E$(BS) are decreased by their respective variance values $V$(IBR) and $V$(BS), and *iii*) *biased high*, where both $E$(IBR) and $E$(BS) are increased by their respective variance values $V$(IBR) and $V$(BS). Regardless of the class, a sample is generated as a sequence of BS and IBR values (mixing services) around their expectation with some

additional random variation defined within their variance magnitude. Note that unbiased samples allow measuring KPIs in average cases, which is intended for computing throughput and average latency. On the contrary, biased samples are designed either for measuring additional delays in the absence of queued traffic (*low*) or stressing the connection capacity to compute maximum latency and packet losses (*high*).

It is important to analyze the generated samples at different time resolutions. Assuming *self-similarity*, at coarse resolution (ms scale) traffic can be seen as a sequence of on/off periods of mixed services, whereas, at a finer resolution (μs scale), the sequence of on/off periods can be seen between bursts of differentiated services. This degree of detail is required to generate precise active probe configuration.

From the generated samples, a procedure to adjust and configure bursts of trains of packets in the active probe is required. Thus, a burst in a measurement configuration corresponds to a total (or partial) burst in a sample. Note that, to regularize the length of the bursts of packet trains, a minimum and maximum number of packets can be setup.

### 5.2.3 CURSA-SQ tuning and ML model training

The real measurements for the set of meaningful configurations obtained are used to tune and validate the CURSA-SQ scenario that will be eventually used to generate synthetic data for model training and validation purposes.

CURSA-SQ requires configuring a set of traffic generators and this can be done according to equations (5-2) and (5-3), as proposed in [Ru18]. However, to reproduce by simulation exactly the same sampled scenarios measured by the active probe, CURSA-SQ traffic generation needs to be altered with the deviation introduced in unbiased measurements.

The generated traffic flows are then propagated through a system of continuous queues $Q$ that models the connection, as represented in Fig. 5-3. Let us assume that every queue $q \in Q$ is characterized by a unique and common buffer with capacity $k$ (in bytes) and a server rate $\mu$ (in b/s). Moreover, $q(t)$ represents the queue state, i.e., the number of bytes in the queue at time $t$. From such state, partial KPIs are computed for each individual queue. Then, computing connection KPIs, i.e., throughput and latency measurements between the source and all the destinations, is simply the aggregation of partial KPIs computed in queues, as well as in delay nodes.

Despite of the fidelity of the CURSA-SQ-based simulation setup to represent a real connection, there are two main unknowns that need to be discovered after analyzing real measurements. First, the magnitude of the additional delay to be introduced by delay nodes can be easily computed after analyzing the mean latency obtained for biased low measurements. Second, as measurements are correlated to some traffic behavior at the burst-level, but they are actually propagated packet-by-packet

during measurements, a mismatch between theoretical and measured traffic behavior can exist. To solve this issue, a *correction factor* (multiplier) can be applied to both expectation and variance configured in the generators in order to fit the characteristics of the expectation and variance measured. A good reference for this purpose is to compare the difference between minimum and maximum latency in the simulation and in the experiments for the case of the biased high monitoring samples. The multiplication factors can be easily obtained to correct the deviation between simulation and experimental values. After this tuning operation, unbiased measurements can be used to validate the accuracy of the simulation environment.

Once CURSA-SQ is tuned and the KPIs obtained by simulation match the experimental ones for the measured samples, a large set of synthetic KPI measurements for a wide range of connection loads along the whole time period can be easily obtained. The data is eventually used for producing ML models that allow the customer to estimate KPIs for each connection destination point as a function of the expected traffic mix. In particular, we use ANNs that, given the aforementioned input, return an output vector with estimation of average throughput, average latency, maximum latency, and packet loss (if any).

## 5.3 Active Measurements

In this section, we present the proposed workflow to be carried out when a new packet connection is requested.

The proposed workflow is shown in Fig. 5-6. It starts when an operator requests the deployment of a new NS through the NFVO's Graphical User Interface (GUI), which defines the interconnected VNFs, as well as the specification of the traffic that can be expected and the QoS constraints in terms of throughput and latency (message 0 in Fig. 5-6). That request triggers the set-up of a number of packet connections, which the NFVO requests through the Parent controller's NBI and includes the traffic specification and the QoS constraints (1). Once the requested connectivity is set-up, the Parent controller updates the MDA controller with the connection ID and its attributes (2). Next, the Parent controller creates an IP multicast group that will be used exclusively for running the commissioning tests, and requests the MDA controller to assess whether the configuration of the connection meets the QoS constraints and to model its performance under the specified traffic (3). Upon the reception of that request in the MDA controller, the PILOT application is triggered. PILOT first determines the probes that will be involved in the tests and requests them to join the IP multicast group (4). In addition, PILOT determines the set of tests to be executed, and for every test, it requests the active probe in the source of the connection to run them specifying the composition of the packet trains that the probe needs to generate. The request includes the VLAN ID that has been configured so the active probe can use it for tagging the generated Ethernet frames (5).

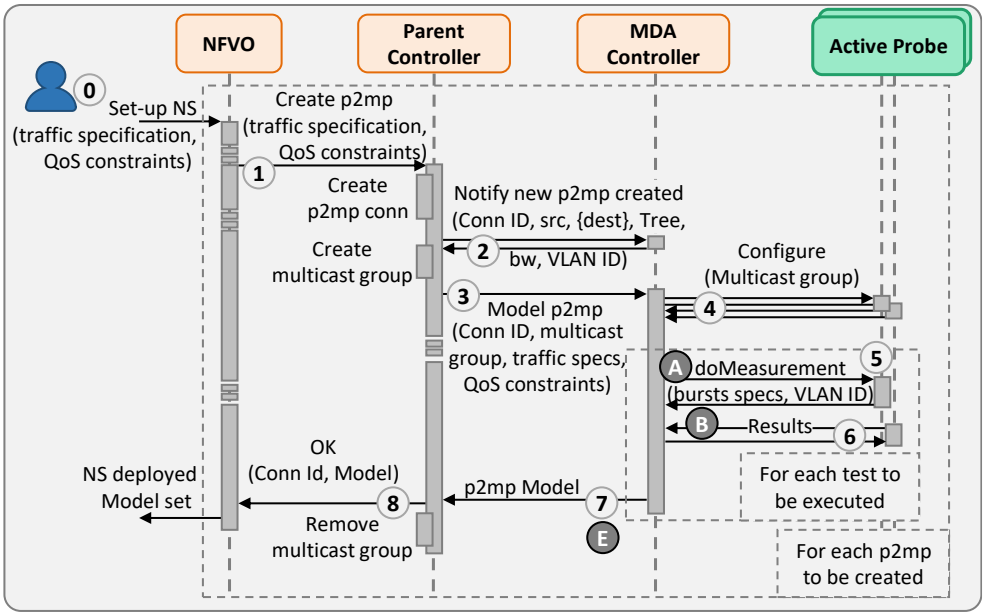


Fig. 5-6. Proposed workflow for a p2mp connection.

To measure the performance of a connection, the active probe in the source CO injects trains of Ethernet frames that are received by the active probe in each remote CO. Once the performance of the connection in every destination CO is measured, the obtained results are notified to the MDA controller (6). With such measurements, the PILOT application configures the scenario and runs CURSA-SQ with specific parameters to generate a training dataset that is subsequently used to generate a ML model for every destination of the connection, as detailed in Section 5.2. New tests are afterwards executed to validate the connection model. Once a ML model for the connection is produced, the MDA controller replies the Parent controller (7), which in turn replies the NFVO (8) and tears down the IP multicast group. The proposed PILOT methodology, the active probes and the workflow are experimentally assessed in the next section.

## 5.4 Experimental Assessment

### 5.4.1 Experimental Platform

Active probes were developed and evaluated locally in a setup in UAM-Naudit premises in Madrid, Spain, (Fig. 5-7a) where the probes were used to evaluate the performance of the connectivity between two L2 Ethernet switches connected through a L3 router with two 1 GbE interfaces.

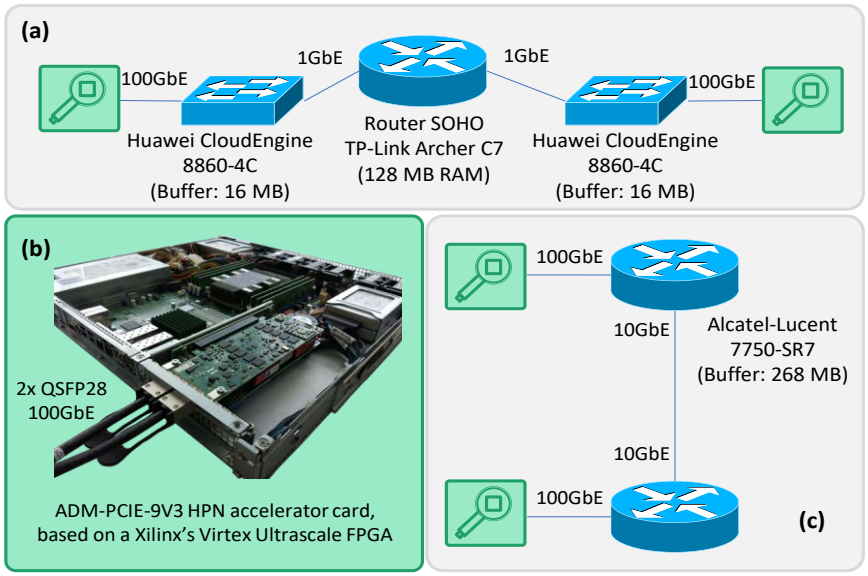


Fig. 5-7. Testbed scenarios and active probes.

5

```
{
 "dst-ip" : "239.10.0.6",
 "measurement-id": 324,
 "vlan-id": 410,
 "allocated-bandwidth-mbps": 1000.0,
 "repetitions": 2,
 "bursts": [{
   "num-packets": 4,
   "packet-size": 1500,
   "delay-till-next-ns": 2000000
  },{
   "num-packets": 6,
   "packet-size": 1500,
   "delay-till-next-ns": 2000000
  },{
   "num-packets": 5,
   "packet-size": 1500,
   "delay-till-next-ns": 2000000
  },{
      :
 }]}
```

6

```
{
 "dst-ip" : "10.10.0.24",
 "measurement-id": 324,
 "mean-packet-size": 1500.0,
 "mean-num-packets": 22.0,
 "throughput-mbps": [41.953, 42.249, 42.545],
 "repetitions": [{
   "repetition-id": 0,
   "packet-loss": 0.000,
   "latency-us": [177.414, 147.634, 110.570],
   "jitter-us": [10.855, 10.027, 8.652],
   "throughput-mbps": [771.57, 740.05, 680.29]
 },{
   "repetition-id": 1,
   "packet-loss": 0.000,
   "latency-us": [162.384, 153.536, 142.635],
   "jitter-us": [11.999, 10.962, 9.222],
   "throughput-mbps": [798.94, 782.18, 775.23]
 }]
}
```

Fig. 5-8. JSON messages for measurement configuration and results.

Fig. 5-8 shows the JSON messages used for measurement configuration and results, which correspond to messages 5 and 6 in the workflow. Message 5 includes the following fields (see Fig. 5-8): *i*) `dst-ip` specifies the IP address of the remote probe (unicast IP address) or probes (the multicast address group created by the Parent controller); *ii*) `measurement-id` uniquely identifies the measurement for correlation purposes; *iii*) `vlan-id` specifies the VLAN tag to be used; *iv*) `allocated-bandwidth-mbps` defines the bandwidth allocated for the connection and is used to define the inter-packet time; *v*) `repetitions` defines the number of times that the list of bursts needs to be sequentially repeated; vi) `bursts` specifies a list of bursts, where each burst is specified by: *a*) `num-packets` defines the number of IP packet to be sent in the burst; *b*) `packet-size` defines the size of each packet in octets; the used BERT

type is Pseudorandom Binary Sequence (PRBS) by default; *c*) `delay-till-next-ns` defines the delay in nanoseconds until the next burst after the end of last packet in the current burst.

With these specifications, the source probe generates the packets for the measurement. Fig. 5-9a shows the generated packets, where one can easily identify the packets trains belonging to the first bursts and the correspondence with the specifications in message 5 in Fig. 5-8. In addition, to graphically illustrate the measurement configured, as well as the effects of the intermediate router, Fig. 5-9b shows the sequence of bursts measured at the output of the source active probe and at the input of the destination one, where some delay and jitter can be observed.

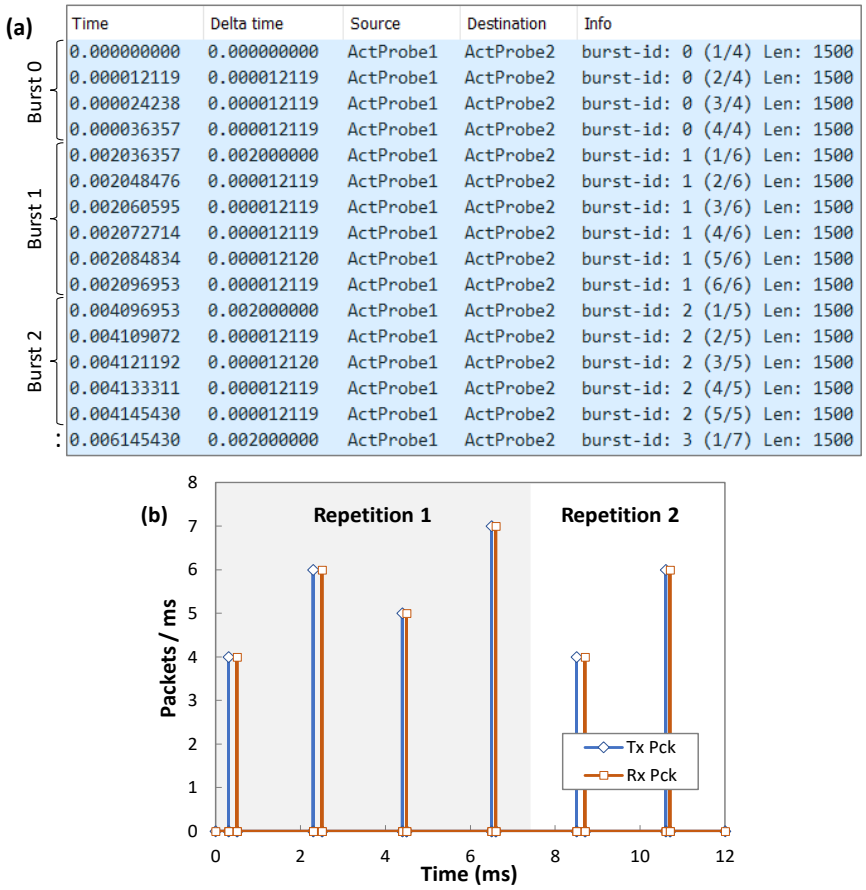


Fig. 5-9. Packets generated for a configured measurement (a) and aggregated generated and received packets (b).

Every probe participating in the measurement returns the results to the MDA controller (6). The message includes (Fig. 5-8): *i*) `dst-ip` is the IP of the probe that reports the results; *ii*) `measurement-id` for correlation purposes; *iii*) `mean-packet-size` is the average size of the packets received; *iv*) `mean-num-packets` is the average number of packets received in every repetition; *v*) `throughput-mbps` specifies the maximum, average and minimum throughput measured for the measurement; and

*vi*) `repetitions` contains a list with the results for each repetition of bursts sent, where each repetition include: *a*) `repetition-id` for correlation purposes; *b*) `packet-loss` reports the packet loss in percentage; *c*) `latency-us`, `jitter-us` and `throughput-mbps` report a list with the maximum, average and minimum latency, jitter, and throughput, respectively, measured during the burst, i.e., excluding inter-burst time, for each repetition independently.

Once the active probes have been assessed, they were deployed in Telefónica premises in Madrid, Spain together with other commercial equipment and the Parent controller. Telefónica premises are connected to UPC ones in Barcelona, Spain, where the MDA controller is deployed. Such distributed testbed is used to carry out the complete experimental assessment and validate the proposed workflow. The setup is shown in Fig. 5-7c, where two active probes were deployed in Telefónica premises and connected to two Alcatel-Lucent routers through 100 Gb/s optical interfaces; the routers are connected through a 10 Gb/s optical link thus creating a virtual link at the packet layer.

### 5.4.2 Methodology validation

The collected measurements were used to generate a relevant set of experimental measurements to tune a CURSA-SQ scenario emulating the real setup. Biased and unbiased samples were generated in the range of normalized load [0.1, 0.95], defined as the average traffic volume over the connection capacity (i.e., 10 Gb/s). We used the statistical service characterization and demand profiles in [Ru18] for generating a mix of different services including Video-On-Demand, Online Gaming, and Internet services.

The results in Fig. 5-10 and Fig. 5-11 show the experimental measurements and the simulation data for two different configurations of the CURSA-SQ scenario, focusing on throughput and latency (both average and maximum) KPIs analysis. Precisely, Fig. 5-10a and Fig. 5-10b show the results obtained from unbiased samples for average analysis, whereas Fig. 5-11 focuses on biased high ones, which are relevant to illustrate the real behavior of maximum latency. In the first CURSA-SQ configuration, simulations have been conducted before tuning CURSA-SQ with the real measurements. The results show a slight reduction of throughput estimation, whereas latency is clearly underestimated due to: *i*) the lack of additional delays consideration, and *ii*) a different latency slope for high loads (clearly visible at loads 0.85 and 0.9). In the second CURSA-SQ configuration, simulations were conducted after tuning CURSA-SQ (according to Section 5.2) using biased low monitoring samples at normalized load 0.1 and biased high ones at normalized load 0.9, for additional delay computation and generators corrections, respectively. As it can be observed, the evolution of all three KPIs as a function of the load closely matches with the experimental measurements.

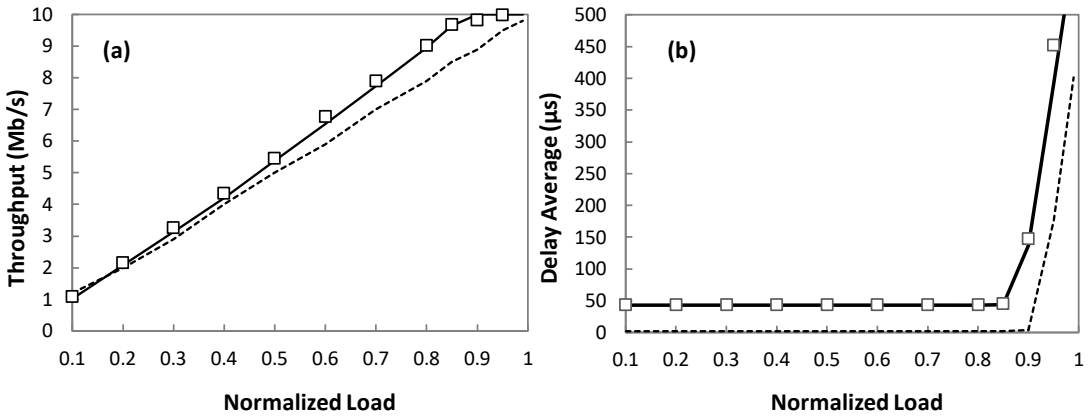


Fig. 5-10. Experimental and simulation results for KPI estimation.

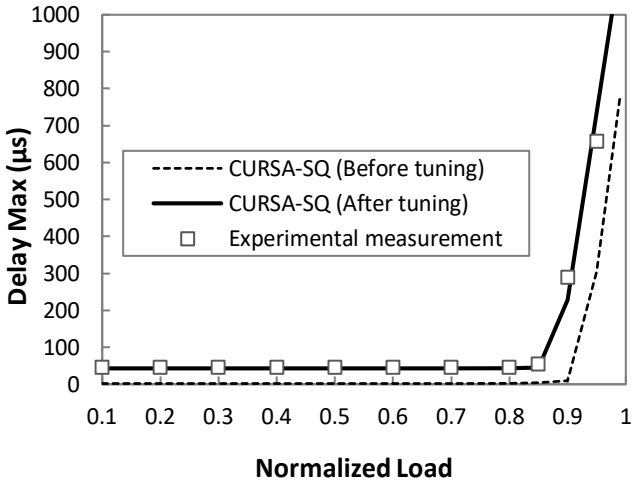


Fig. 5-11. Maximum latency for KPI estimation.

Let us to illustrate the need to tune CURSA-SQ with measurements over packet trains that follow the expected traffic (as detailed in Section 5.2) instead of over packet trains generically configured, i.e., independent to the expected traffic. In the latter, the active probes can generate two types of trains: *i*) with small packets and large inter-packet time to measure minimum latency; and *ii*) with large packets and small inter-packet time to measure maximum throughput [Ru16.2]. Fig. 5-12 compares the performance at the highest loads of tuning CURSA-SQ based on real measurements over both active probes configurations, for throughput (a), average delay (b) and maximum delay (c). The results show a sub-estimation of the delay as large as 70%. In view of the results, we can conclude that the proposed packet trains configuration results in precise measurements and thus better CURSA-SQ tuning.

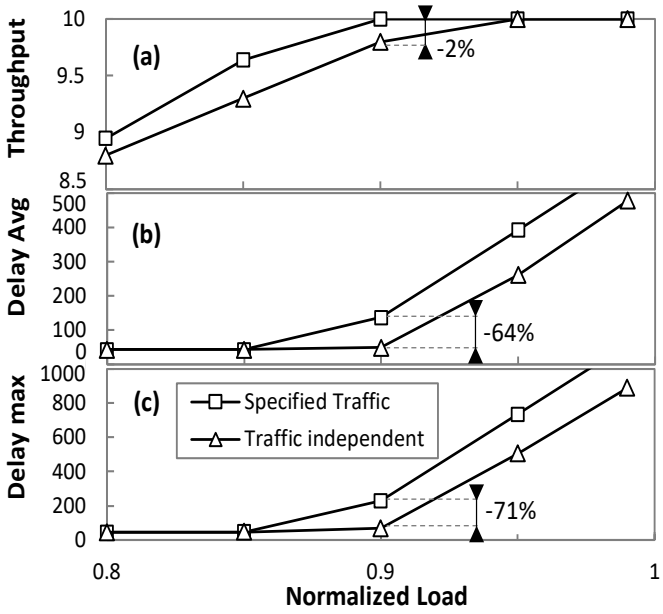


Fig. 5-12. CURSA-SQ tuning as a function of the injected packet trains.

An additional analysis of the gain introduced by the proposed tuning methodology is depicted in Fig. 5-13, where the relative errors between simulated and real KPIs are computed for both CURSA-SQ configurations. It is worth noting that the remarkable reduction of error achieved by the tuning procedure. In fact, error below 3% for estimated average and 9% for maximum latency are obtained, which validates the simulation environment as accurate synthetic monitoring data generator for KPI model training purposes.

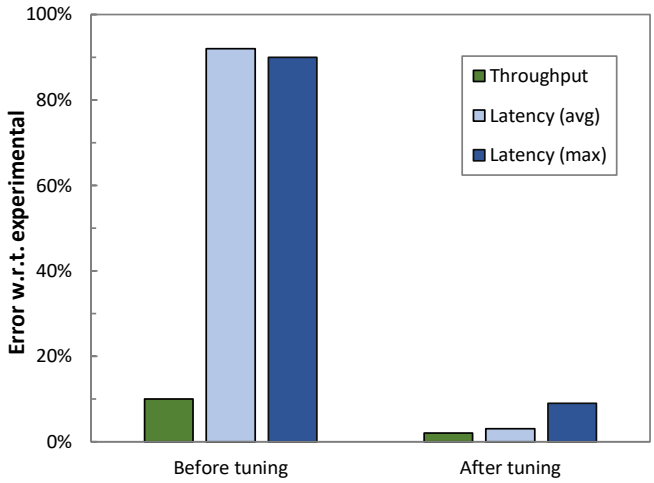


Fig. 5-13. Relative error of CURSA-SQ-based simulation.

### 5.4.3 Real vs Synthetic data for ML training

Once the simulation environment has been validated, let us study the benefits of using PILOT with CURSA-SQ tuned with real measurements sampling the specified traffic mix, as compared to using just real measurements to generate labeled data for ML training and validation. To this end, we conducted a simulation-based study in a more complex scenario, according to the scheme represented in Fig. 5-3, where the delay nodes (d) implement a non-linear function that follows the behavior found in the previous subsection.

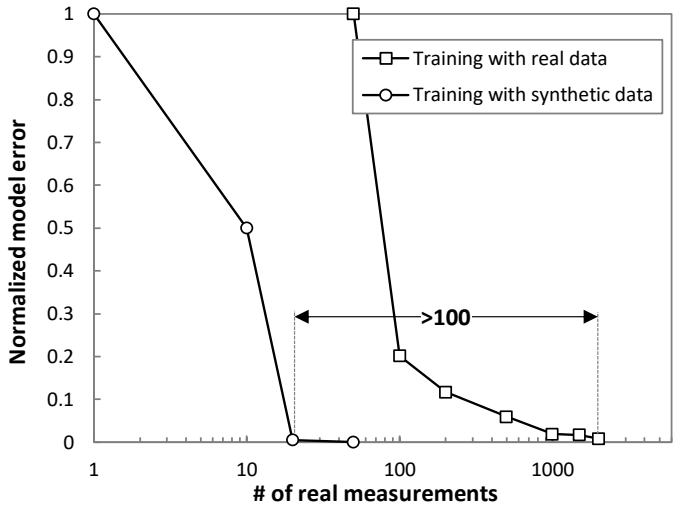


Fig. 5-14. Prediction error of ML models vs # of real measurements.

Fig. 5-14 shows the prediction error (normalized between the minimum and maximum) of the trained ML models as a function of the number of real measurements conducted. Note that the approach based on synthetic samples has been configured to perform ML training using 10,000 training data samples regardless of the number of real measurements used for the initial tuning. On the contrary, in the approach of ML training with real data, the number of training data samples equals that of real measurements. In both cases, an ANN with a single input for the normalized NS load and one output for each KPI was trained (Section 5.2.3). A single hidden layer consisting in 10 hidden neurons with the logistic activation function was configured. The training procedure converged to accurate ANNs in less than one minute in a conventional desktop computer, which points out a negligible computational burden. The results clearly show the benefits of using the proposed PILOT methodology: the minimum model error is achieved with less than 100 times of experimental measurements as compared with the training with real data. In fact, note that the minimum error with the PILOT methodology is achieved when the approach based on real measurements has not yet enough data to initiate ANN training. Therefore, we can conclude that using CURSA-SQ tightened with relevant

active measurements allows obtaining accurate KPI models for packet connections during commissioning testing.

## 5.5 Concluding Remarks

A methodology named PILOT has been proposed and experimentally demonstrated to provide predictable connectivity services. The PILOT methodology facilitates reducing the cost of overprovisioning at both, the packet and the optical layer. PILOT is based on three main pillars that allow the generation of accurate ML models to estimate the QoS, in terms of throughput and latency, during commissioning testing: *i*) an efficient traffic flow simulation environment, named CURSA-SQ, to produce large amounts of labeled data for ML training and validation purposes; *ii*) real measurements to tune the CURSA-SQ scenario by discovering additional delay and throughput bottlenecks, which are usually not constant but related to the actual traffic load; and *iii*) specification of the estimated traffic mix that the connection will support are used in the process of data generation, which includes real measurements and traffic generation for the simulation scenario.

PILOT is carried out during the provisioning phase of NSs as part of commissioning testing. In this regard, a workflow has been proposed that executes PILOT for every connection related to the NS being provisioned. As a result, a ML model is produced for every connection and the set of models are returned to the client at the end of the provisioning phase.

The active probes were first experimentally validated in a local setup and then integrated in a distributed testbed connecting UPC and Telefónica premises, respectively, in Barcelona and Madrid, Spain, where the PILOT methodology and the proposed workflow were experimentally assessed. The results show noticeable accuracy of the produced ML models after the scenario in CURSA-SQ was tuned with real measurements. Last but not least, the benefits of using PILOT with CURSA-SQ were compared to using just real measurements to generate labeled data for ML training and validation in terms of number of real measurements needed to train accurate ML models. The results show that PILOT requires around 20 real measurements, which is 100 times less than performing ML training directly with real measurements.

Many of the concepts developed in this chapter are used in the next one for multi-domain scenarios, where each single domain can model intra-domain connection delay as defined here.

# Chapter 6

# Delay Modeling in Multi-Domain Networks

Accurate delay estimation is one of the enablers of future network connectivity services, as it facilitates the application layer to anticipate network performance. If such connectivity services require isolation (slicing), such delay estimation should not be limited to a maximum value defined in the Service Level Agreement, but to a finer-grained description of the expected delay in the form of, e.g., a continuous function of the load. Obtaining accurate end-to-end (e2e) delay modeling is even more challenging in a multi-operator (Multi-AS) scenario, where the provisioning of e2e connectivity services is provided across heterogeneous multi-operator (Multi-AS or just domains) networks.

In this chapter, we propose a collaborative environment, where each domain SDN controller models intra-domain delay components of inter-domain paths and share those models with a broker system providing the e2e connectivity services. The broker, in turn, models the delay of inter-domain links based on e2e monitoring and the received intra-domain models. Exhaustive simulation results show that composing e2e models as the summation of intra-domain network and inter-domain link delay models provides many benefits and increasing performance over the models obtained from e2e measurements.

## 6.1 Motivation and objectives

In this chapter, we present a novel approach for QoS modeling in multidomain networks. The approach is built on top of the principle of cooperative learning, where different entities establish a positive interdependence for the sake of a powerful

collective intelligence. We propose a collaborative approach between the domains and the broker, where the domains exchange abstracted data to achieve robust and accurate ML models for e2e QoS (delay) estimation. Those models can then be used for multiple purposes, such as connectivity provisioning based on QoS estimation or connectivity reconfiguration triggered by anticipated detection of QoS degradation. Assuming that each network domain can model its internal, limited view of delay based on its own traffic and delay measurements, the broker entity plays the role of combining the different domain delay models for an e2e view. By providing models instead of data, domains can share knowledge with the broker and, at the same time, enforce privacy policies. With this approach, the broker can detect and infer inaccuracies in the partial models to trigger their retraining. Specifically, the contributions of this chapter are:

- The collaborative approach between the domains and the broker, where the latter provides multidomain connectivity with QoS constraints to end customers. Domains share delay models with the broker, which composes e2e delay models as combination of domain and inter-domain link models. Such approach enables the capacity to infer delay performance before establishing an e2e path, as well as to find the cause of deviations between the model and the measurements.
- The specific procedures for: i) inter-domain link delay estimation based on e2e delay measurements and per domain delay models; ii) intra-domain model inaccuracy detection with measures to correct intra domain models; iii) detection of the source of the inaccuracy once it is detected.

## 6.2 End-to-end and per-domain delay estimation

Fig. 6-1 shows the control architecture considered in this chapter. We assume a dynamic scenario where a connectivity manager in the broker receives and processes requests for connectivity with QoS requirements in terms of throughput and maximum delay among customer endpoints. The *connectivity manager* uses a *planning tool* for provisioning and reconfiguration purposes [Ve17.2], and it is assisted by ML- based models to enhance connectivity provisioning. The broker connects to a set $D$ of domains interconnected by a set L of inter-domain links and altogether provides connectivity to a set $P$ of e2e connections, the performance of which is continuously monitored.

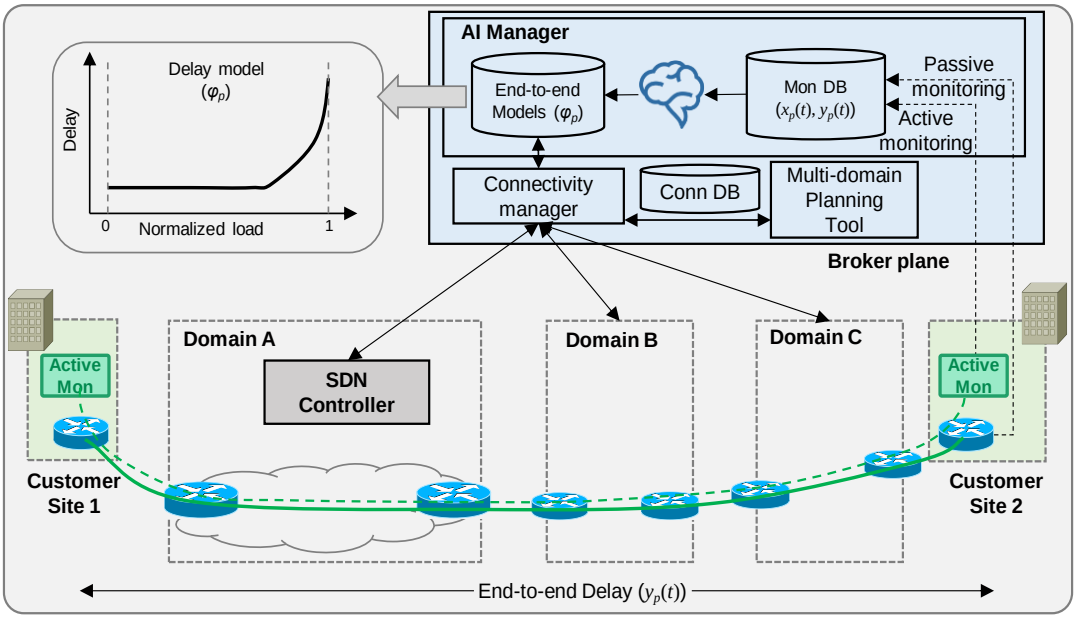


Fig. 6-1. Example of e2e delay and control architecture.

Let us assume first that the performance of each connection p is monitored e2e between the endpoints in the customer sites and that data is gathered by the broker for various purposes, like QoS analysis, modeling, etc. In particular, we assume that throughput, $x_p(t)$, and delay, $y_p(t)$, are measured periodically, e.g., every 1 min, by the customer edge routers (passive monitoring) and that active monitoring is carried out. Then, after a sufficiently large period, enough data can be collected to train ML models for every connection. Note that protocols like IPFIX, gRPC, etc. can be used for monitoring data collection.

Among possible ML models, path delay models (denoted as $\varphi^*_p$) can be used to predict a delay-related performance metric, e.g., average or maximum delay, as a function of the normalized load (computed as the ratio between the measured throughput and the capacity of the path) (see Chapter 5). The embedded graph in Fig. 6-1 illustrates an example of delay model. End-to-end delay ML models can be used, e.g., to anticipate QoS degradation and trigger reconfiguration.

Note that the $\varphi^*_p$ models not only allow analyzing the e2e delay of a single path p but they can also be used to get some insight on the performance of the domains by considering groups of paths that cross a given domain. An example is in the case of detecting model inaccuracies (e.g., significantly higher delay than expected for the observed traffic load) in a group of paths; correlation of their routes through the domains can lead to finding a common set of *segments*, either domains or inter-domain links, that could potentially hold the source of the inaccuracy. Once some segment(s) have been identified, re-routing of those affected paths could be performed to avoid them.

Nevertheless, this AI-assisted architecture suffers from an inherent drawback: the multidomain network is analyzed as a black-box, a fact that limits the applicability of advanced multidomain smart operation. An example is illustrated in Fig. 6-2, where two paths between domains A and B (p1 from *RA.x* to *RB.y* and *p2* from *RA.y* to *RB.x*) are established and e2e delay models have been accurately trained after some data collection phase (Fig. 6-2a). Then, let us now consider that a new request for a path from *RA.x* to *RB.x* arrives. Since no path between *RA.x* and *RB.x* was previously established, the broker does not have available e2e monitoring data and delay models to predict the delay behavior of such new connectivity request. This fact limits smart provisioning decision making, e.g., to choose the best route in terms of QoS.

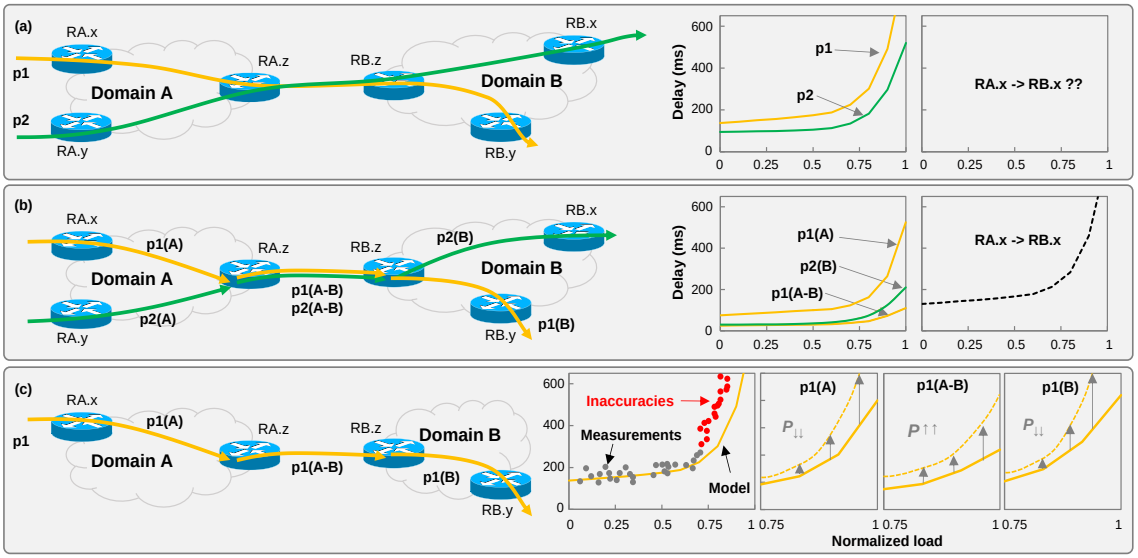


Fig. 6-2. Provisioning of multidomain requests: reference approach (a) vs compound approach (b). Example of inaccuracy (c).

To overcome the aforementioned issue, we propose breaking the black-box, monolithic view of the multidomain network. End-to-end delay can be modeled by combining intra-domain and inter-domain segment models for those segments in the route of a path. This brings some benefits, as segment models can be used to create compound models not only for those established paths but also to infer models for not yet established paths. Following the previous example, the inferred model for the path request between *RA.x* to *RB.x* could be obtained by composing an e2e delay model from segments models: *p1(A)*, *p1(A-B)*, and *p2(B)* (Fig. 6-2b). Note that intra-domain segment models need to be computed by the domains themselves based on, e.g., active measurements carried out between the access/inter-domain interfaces for a given path in the domain. Besides, the delay introduced in inter-domain links needs also be measured, which is more difficult as active measurements should be carried out across domains. For this very reason, we target at modeling the delay of

inter-domain links at the broker level thus relaxing the need of multidomain active monitoring.

Let us see now how we can identify the segment(s) that are producing delays higher than expected. Once inaccuracies between model predictions and real measurements are detected for these segments, they can be excluded for route computation in case re-routing needs to be performed. Note that the black-box approach presented above identifies common segments of affected paths by correlating routes. However, such a procedure would be imprecise and even meaningless if inaccuracies are detected in only few paths. Fig. 6-2c illustrates an extreme case, where inaccuracies are observed in only one path for high loads. By analyzing the models of every segment and computing how likely it is that each segment introduced such inaccuracy the most reasonable segment can be selected. With this narrower identification, more precise and proper reconfiguration actions can be taken.

The extended architecture is presented in Fig. 6-3, where only the details of the AI-related components are represented for the sake of clarity. Compound e2e delay models are composed as a combination of two distinct types of models: i) *intra-domain models ($\varphi_{pd}$)* and *ii) inter-domain link models ($\varphi_l$)*. The intra-domain models are obtained by each domain controller from delay and throughput measurements collected for each of the e2e path segments carried by that domain.

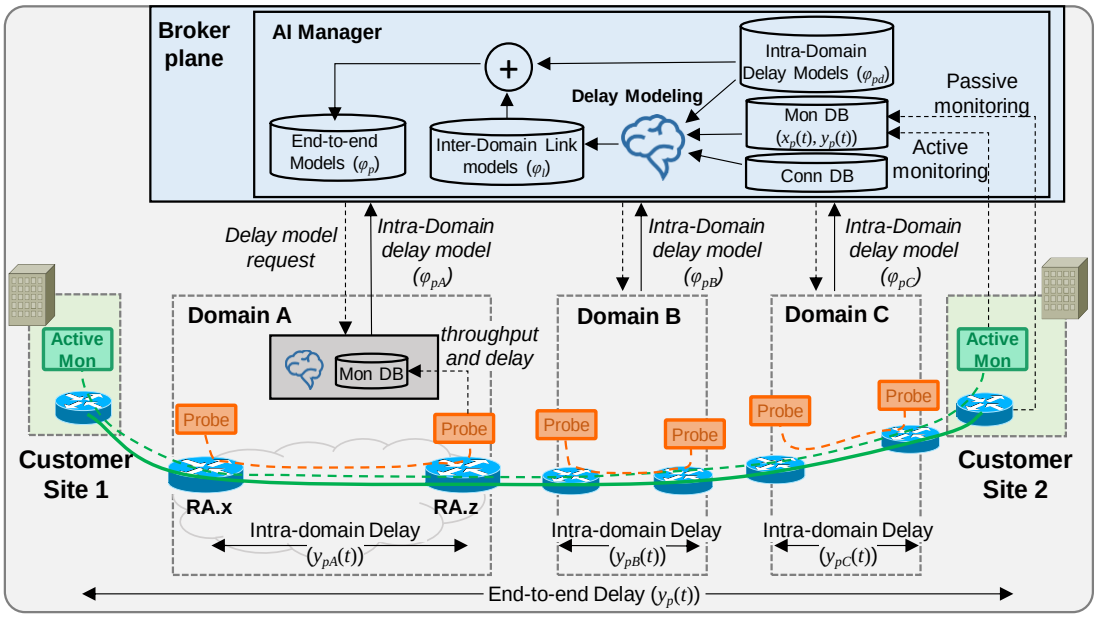


Fig. 6-3. Example of per-domain e2e delay and extended control architecture.

As an example, Fig. 6-3 represents an e2e path p that crosses domain A from node *RA.x* to *RA.z*; therefore, the manager in domain A finds the model that predicts the delay that the e2e path experiences when crossing domain *A ($\varphi_{pA}$)*. To enforce domain privacy, domain models make predictions without the required knowledge from the actual domain state (e.g., the actual routes of the paths). The inter-domain models

are trained by the broker, with the input of the e2e measurements and the predictions of the domain models, so that the inter-domain link components ($\varphi_l$) can be inferred. Note that, under this hierarchical training architecture, every entity (domains and broker) is free to use their modeling techniques bringing the best tradeoff between complexity and accuracy in their domains. Therefore, the combination of models into an e2e delay model needs to be done assuring that different types of models can co-exist, assuming that all delay components are functions of traffic load.

## 6.3 Compound e2e delay modeling

This section details the processes that together provide accurate compound e2e delay models. Table 6-1 introduces the notation and defines the main parameters and variables that will be consistently used hereafter. Besides, as introduced in Section 6.2, we define the compound e2e delay model of a given path $p$ as the sum of their intra-domain and inter-domain link components, which can be formally expressed as follows:

$$\varphi_p\left(x_p(t)\right) = \sum_{d \in D} \varphi_{pd}\left(x_p(t)\right) + \sum_{l \in L} \delta_{pl} \cdot \varphi_l\left(x_l(t)\right) \tag{6-1}$$

*Table 6-1: Notation*

| *Sets and parameters* | |
|---|---|
| $P$ | Set of multidomain paths, index $p$. |
| $X=\{x_p\}$ | Set of e2e traffic monitoring samples for all paths. |
| $Y=\{y_p\}$ | Set of e2e delay monitoring samples for all paths. |
| $D$ | Set of domains, index $d$. |
| $L$ | Set of inter-domain links, index $l$. |
| $P(d)$ | Subset of paths that traverse domain $d$ |
| $\delta_{pl}$ | 1 if path $p$ traverses inter-domain link $l$ and 0 otherwise. |
| $x_i(t)$ | Measured traffic throughput through element $i$ at time $t$, where $i$ might identify a path $p$ or a link $l$. |
| $\varphi_p$ | Compound e2e delay model for path $p$ |
| $\varphi_{pd}(x)$ | Intra-domain delay model for path $p$ in domain $d$. The zero function if $p$ does not cross $d$. |
| $\varphi^*_d$ | Correction model for intra-domain delay for domain $d$. |
| $\varphi_l$ | Inter-domain link delay model for link $l$ |

| | |
|---|---|
| $Ttr$ | Training phase duration (in monitoring intervals) |
| $y_p^{inter}(t)$ | Aggregated inter-domain link delay of path $p$ at time $t$ |
| $E[\cdot]$ | Expectation |
| *Decision Variables* | |
| $y_l(t)$ | Delay at inter-domain link $l$ at time $t$ |
| $\beta_{(\cdot)}(t)$ | Delay bias of path/domain |
| $u_p(t)$, $v_p(t)$ | Delay slack and surplus variables for path $p$ |

Fig. 6-4 presents the main building blocks and their relationships for compound e2e delay modelling at the broker plane. Blocks have been conveniently numbered to facilitate the ongoing description and workflows. As depicted in Fig. 6-4, the main blocks can be organized into three clearly differentiated groups: *i)* inter-domain link delay component estimation (blocks 1-3); *ii)* inter-domain link delay model training (blocks 6a and 7a);and *iii)* intra-domain delay model correction (blocks 7b and 8b), which also includes inaccuracy detection (block 5) and localization procedure (block 6b) that keeps the highest goodness-of-fit through precise model improvement actions. In turn, some of the blocks perform some computation or solve optimization problems.

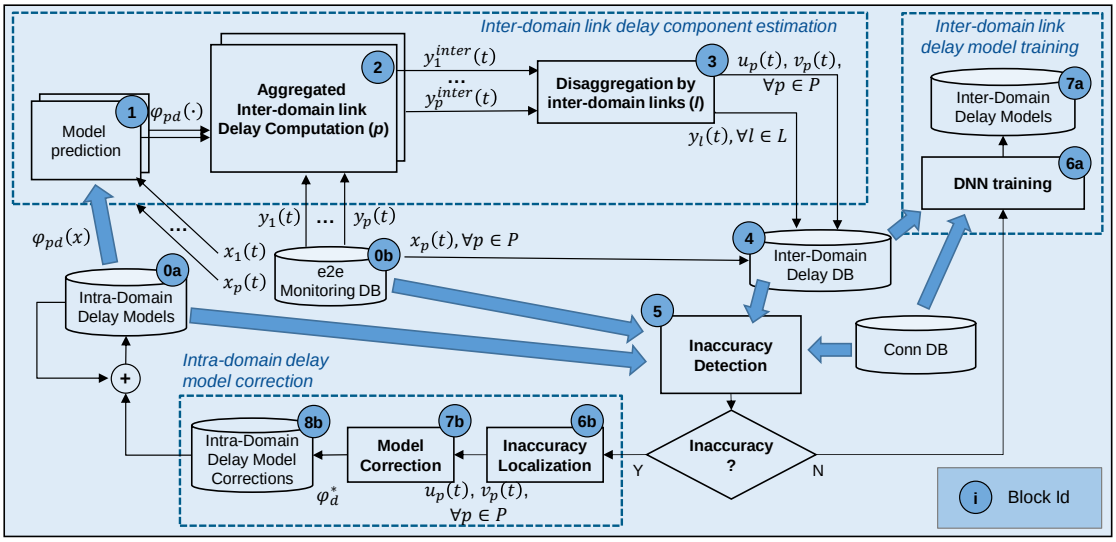


Fig. 6-4. Main building blocks for training and correcting delay models at the broker plane.

For the sake of clarity, Algorithm 6-1 presents the pseudocode for inter-domain link delay modeling and intra-domain model correction, and Table 6-2 relates the defined blocks to optimization problems or equations. The details of the above groups and blocks are presented in the next subsections.

*Algorithm 6-1: Inter-Domain Link Delay Modeling with Intra-Domain Model Correction*

**Input:** $X$, $Y$, $\{\varphi_{pd}\}$

**Output:** $\{\varphi_l\}$, $\{\varphi^*_d\}$

```
Corrections ← {}
while true
  {y_p^inter}← Inter-domain_Delay_Estimation(X, Y, {φ_pd},
       Corrections) (block 2)
  {y_l},{u_p},{v_p} ← Link_Delay_Disaggregation_Bias ({y_p^inter},
            P, L) (block 3)
  inac ← Inaccuracy_Detection({u_p}, {v_p}) (block 5)
  if !inac then
    {φ_l}←DNN_training (X, {y_l}, P) (block 6a)
    break
  d*,{u_p},{v_p}←Inaccuracy_Localization({y_p^inter},P,L) (block 6b)
  φ*_d ← Model_Correction (d*,{u_p}, {v_p}) (block 7b)
  Corrections ←Corrections ∪ {φ*_d}
return {φ_l}, Corrections
```

*Table 6-2: Relation Between Blocks and Problems/Eqs*

| Block | Problem | Eq(s). |
|---|---|---|
| 2 | *Inter-domain Delay Estimation* | (6-2) |
| 3 | *Link Delay Disaggregation* | (6-4)-(6-5) |
| 3 | *Link Delay Disaggregation Bias* | (6-7)-(6-10) |
| 5 | *Inaccuracy Detection* | (6-11)-(6-12) |
| 6b | *Inaccuracy Localization* | (6-5)-(6-8)-(6-9)-(6-13) |

### 6.3.1 Inter-domain link delay modeling

For modeling inter-domain link delay components, a training database (DB) with inter-domain link delays ($y_l$(t)) is constructed based on the inter-domain link delays estimation, given intra-domain delay models and the measured throughput and e2e delay for the paths. We assume that the intra-domain delay models have been received from the domain controllers and are stored in a DB, and a meaningful phase of monitoring data collection spanning *Ttr* monitoring time periods has been carried out and the data are stored in an e2e monitoring DB (this phase concerns blocks labeled 0a and 0b in Fig. 6-4); the proper value of *Ttr* needs to be chosen considering the trade-off between the required sample size to obtain meaningful inter-domain link models and the time needed to collect all monitoring measurements.

Once data are available, the delay component estimation starts, and for each collected measurement $<x_p(t), y_p(t)>$, $t$=1..$Ttr$, several steps are executed to infer the components of such delay introduced for each inter-domain link crossed by a path. First, the domain delay models are used to produce the delay expected ($\varphi_{pd}(\cdot)$) in every domain (block 1). Next, block 2 focuses on isolating the per-path aggregated inter-domain delay component $y_p^{inter}(t)$, defined as the remainder of delay that cannot be explained by the summation of the expected domain contributions predicted by domain models; $y_p^{inter}(t)$ can be formally defined as:

$$y_p^{inter}(t) = y_p(t) - \sum_{d \in D} \varphi_{pd}\left(x_p(t)\right) \tag{6-2}$$

Consecutively, block 3 processes jointly all $y_p^{inter}(t)$ values to disaggregate the delay per-inter-domain link. The result of this step allows inferring inter-domain link delays $y_l(t)$ from monitoring data. This inference is supported by the assumption that the expectation ($E[\cdot]$) of per-path aggregated inter-domain delay component equals the sum of the expectations of the delays introduced by each inter-domain links of the path, i.e.:

$$E\left[y_p^{inter}(t)\right] = \sum_{l \in L} \delta_{pl} \cdot E[y_l(t)], \quad \forall p \in P \tag{6-3}$$

According to equation (6-3), the estimation of $y_l(t)$ values given a set of paths $P$ and a set of per-path aggregated inter-domain delays $y_p^{inter}(t)$ can be done by simple regression techniques. In this work, we propose implementing the disaggregation block (3) by using the least absolute deviation regression [Ea11], which entails solving the *Link Delay Disaggregation* optimization problem in equations (6-4)-(6-5) independently for each $t$=1..$Ttr$:

$$\min \sum_{p \in P} \left| y_p^{inter}(t) - \sum_{l \in L} \delta_{pl} \cdot y_l(t) \right| \tag{6-4}$$

subject to:

$$y_l(t) \in \mathbb{R}^+, \quad \forall l \in \mathrm{L} \tag{6-5}$$

After solving the above optimization problem, we apply spline smoothing to the obtained $y_l(t)$ values to make them more consistent with the continuous temporal collection and to eliminate those variations resulting from solving each time independently. The results are then used to populate a training dataset (block 4), together with the model input features, i.e., the measurements of the e2e traffic $x_p(t)$.

The resulting dataset can be used for training a fully connected, feedforward DNN (block 6a) that predicts $\varphi_l$ of every inter-domain link as a function of both the traffic $\{x_p(t)\}$ and the route (only inter-domain links) of the paths ($\{\delta_{pl}\}$). The DNN exploits the fact that different paths crossing different inter-domain links could have, however, similar behavior and correlation between traffic and delay. The trained models are stored in a DB and are ready to be used (block 7a).

### 6.3.2 Intra-domain model correction

Although the procedure in the previous subsection has been designed to achieve accurate estimation of the actual inter-domain link delays, there are two cases where that accuracy can be seriously affected: *i*) the availability of a limited number of multidomain paths with few distinct routes can lead to the impossibility of properly isolating and inferring inter-domain link delays. In this regard, our proposed method exploits as much as possible the available information from existing multidomain paths to produce the most accurate compound e2e delay models; *ii*) inaccurate intra-domain delay models. Note that those models are obtained during the commissioning testing phase and updated periodically using active probes, which, as discussed in the introduction, need to be properly configured as otherwise, delay measurements could largely differ from those experienced by the real traffic, thus resulting in inaccurate intra-domain delay modeling.

Especially for the second case, the broker can play a key role in detecting, identifying, and correcting the intra-domain delay model inaccuracies before compound models are used. Note that the benefits are two-fold: 1) after intra-domain models are properly corrected, the broker can make use of accurate compound e2e models without any re-training performed by domains; and 2) the applied corrections can be notified to the affected domain(s), which in turn can use that useful information to tune and adapt its/their mechanism for intra-domain modeling of future services, e.g., using more realistic packet trains used by the active probes.

Before introducing the procedure to detect and identify intra-domain delay model inaccuracies and compute model corrections, the formulation proposed in Section 6.3.1 needs to be revisited.

The presence of inaccuracies in the intra-domain delay models impacts negatively on the veracity of the assumption formulated in equation (6-2) and now $y_p^{inter}(t)$ values contain not only the aggregated inter-domain link delay component but also the error (underestimation or overestimation) introduced by inaccurate intra-domain delay models. Since inter-domain links can support both accurate and inaccurate paths, finding a common inter-domain link delay value that fits all the paths traversing the link is, by definition, imprecise. In other words, the expression in equation (6-3) defines an expectation of inter-domain link delay that could be far from the true value. Consequently, the condition in equation (6-3) need to be extended to incorporate a *per-path bias* $\beta_p(t)$ that collects those potential intra-domain inaccuracies:

$$E\left[y_p^{inter}(t)\right] = \sum_{l \in L} \delta_{pl} \cdot E[y_l(t)] + \beta_p(t), \quad \forall p \in P \tag{6-6}$$

Hence, the *Link Delay Disaggregation* optimization problem in equations (6-4)-(6-5) needs to be extended to quantify that bias for every path; the mentioned optimization is reformulated as follows:

$$\min \sum_{p \in P} \beta_p(t) \tag{6-7}$$

subject to:

$$y_p^{inter}(t) = \sum_{l \in L(p)} \delta_{pl} \cdot y_l(t) + u_p(t) - v_p(t), \qquad \forall p \in P \tag{6-8}$$

$$\beta_p(t) = u_p(t) + v_p(t) \tag{6-9}$$

$$y_l(t) \in \mathbb{R}^+, \quad \forall l \in \mathrm{L} \tag{6-10}$$

The *Link Delay Disaggregation Bias* optimization problem—that now implements block 3—finds the least absolute deviation of inter-domain link delay components in equation (6-8) with the adjustment of both slack and surplus variables for each path and time $t$ ($u_p(t)$ and $v_p(t)$). Equation (6-9) relates per-path bias to slack and surplus variables). Note that as the *Link Delay Disaggregation* problem, the *Link Delay Disaggregation Bias* one needs to be solved for all samples collected during the training period defined by *Ttr*. However, this problem produces not only the set of all inter-domain link delay components but also the set of slack and surplus values to be stored in the inter-domain delay DB (block 4).

Per-path slack and surplus values are analyzed in the inter-domain delay validation (block 5) by solving the Inaccuracy Detection problem. This problem aims at identifying the presence of a large bias as a consequence of some intra-domain delay model inaccuracies. Specifically, a decision score s is defined based on key statistical quartiles [Ba15] of the average bias of every path in time. Equation (6-11) formally describes the computation of the quartiles 25%, 75%, and 100% % of the bias of all paths. The obtained results are then used to compute s in equation (6-12) where the interquartile range ($q_{75\%} - q_{25\%}$) is multiplied by the maximum $q_{100\%}$.

$$\langle q_{25\%}, q_{75\%}, q_{100\%} \rangle = Q\left(\frac{1}{T} \cdot \sum\nolimits_{t=1..Ttr} \beta_p(t)\,,\ \forall p \in P;\ \langle 25\%, 75\%, 100\% \rangle\right) \tag{6-11}$$

$$s = (q_{75\%} - q_{25\%}) \cdot q_{100\%} \tag{6-12}$$

Intra-domain model inaccuracies increase the bias of some paths, so we expect that both maximum $q_{100\%}$ and interquartile range ($q_{75\%} - q_{25\%}$) increase, which makes that the proposed score sharply increases. In the case that the score is under a predefined threshold, then the inter-domain components in the dataset (block 4) are validated and they can be used for training the DNN (block 6a); otherwise, the phase of inaccuracy localization starts.

### 6.3.3 Inaccuracy localization

Upon the detection of inaccuracy, the localization of the source of such inaccuracy (block 6b) can be done by solving the *Inaccuracy Localization* optimization problem, which is a variation of the *Link Delay Disaggregation Bias* one. This variation

requires selecting one domain $d$ at a time and the set of paths crossing it. The formulation of the *Inaccuracy Localization* problem is as follows:

$$\min \beta_d(t) = \frac{1}{|P \backslash P(d)|} \cdot \sum_{p \in P \backslash P(d)} \beta_p(t) \tag{6-13}$$

subject to: Constraints (6-5), (6-8) and (6-9).

The *Inaccuracy Localization* problem excludes domain $d$ from the objective function and therefore, slack and surplus variables of the paths traversing $d$ can take any value with no additional cost. Then, if the inter-domain link delays can be obtained without significant bias of the non-affected paths, the selected domain is a source of inaccuracy. Therefore, we solve the *Inaccuracy Localization* problem for every domain and select the one with the lowest bias $\beta_d(t)$ as responsible for the inaccuracy.

Finally, block 7b estimates—e.g., by applying cubic spline regression [Kn00]—the needed correction $\varphi^*_{pd}$ as a function of the load using the obtained slack and surplus values. Such corrections are stored in a DB (block 8b), so the prediction of intra-domain models is computed as the sum of the prediction of the model itself plus the prediction of the correction model.

## 6.4 Illustrative Results

This section presents simulation results to validate the blocks, models, and procedures described in Section 6.3.

### 6.4.1 Simulation Scenario

CURSA-SQ (see Section 2.3) was used as a simulation environment. The CURSA-SQ methodology was tuned with the experimental measurements presented in Chapter 5, and successfully used to reproduce realistic packet scenarios, including converged fixed-mobile [Be20] and time-sensitive networks [Ve20].

A multidomain topology was reproduced, which consisted of four domains and 12 inter-domain unidirectional links connecting border routers of two different domains. Seven customer sites (endpoints of multidomain paths) were connected to domain edge routers in every domain. Five distinct candidate routes ranging between 500 km and 1000 km were considered for every pair edge-border and border-border routers, to emulate a wide variety of intra-domain networks. The topology represents a moderated but realistic size of a multidomain topology [Ch18]. Based on this reference topology, scenarios consisting of several multidomain paths traversing 3 out of 4 domains, were generated. Every multidomain path requested 10 Gb/s maximum capacity and was routed randomly, firstly at inter-domain level (by selecting a sequence of domains and inter-domain links) and secondly at intra-domain level (by selecting a candidate route per each path segment), being the total

length of the e2e routes between 1,000 and 3,000 km. Inter-domain links were dimensioned to fit the sum of maximum traffic of all traversing paths. Besides, background single domain 10 Gb/s paths, using the same candidate intra-domain routes, were added to generate different scenarios, where the proportion of delay introduced by inter-domain links with respect to the total e2e delay (hereafter, denoted as proportion ρ) varies. CURSA-SQ was used to generate seven days of traffic, emulating daily variations under the assumption of stationary traffic conditions for each configuration: <number of multidomain paths, $\rho$>. The generated traffic was injected in the scenario defined above. The characteristics of the traffic are summarized in Table 6-3.

*Table 6-3: Characteristics of Generated Traffic*

| | **min** | **max** | **median** | **mean** | **std. dev.** |
|---|---|---|---|---|---|
| **Traffic (Gb/s)** | 0.78 | 10 | 3.68 | 3.95 | 2.56 |
| **Delay (ms)** | 9.42 | 75.38 | 13.24 | 19.32 | 12.92 |

The next subsections present the obtained results.

### 6.4.2 Inter-domain link delay modeling

Let us first study and validate the accuracy of the proposed methodology for inter-domain link delay estimation, i.e., blocks 2 and 3 in Fig. 6-4 including the *Link Delay Disaggregation* problem defined in Section 6.3.1, assuming the availability of accurate intra-domain delay models. Since we simulated each delay component to elaborate e2e measurements, we had access to the *real* delay introduced by each inter-domain link. Therefore, we compared the estimated delay against the real one and computed the inter-domain link delay error estimation.

Fig. 6-5 presents the obtained results as a function of two of the most relevant factors impacting on such error: *i*) the number of multidomain paths; and *ii*) parameter $\rho$. In particular, Fig. 6-5a plots the performance for several values of ρ as a function of the number of paths, whereas Fig. 6-5b highlights the results for 60, 120, and 240 paths as a function of $\rho$. We observe that the higher the number of multidomain paths is, the more accurate the delay estimation becomes. However, the proportion of multidomain traffic is crucial to determine the magnitude of this error. In scenarios where the intra-domain delay is far greater than the inter-domain one ($\rho$=15%), achieving the desired target error below 5% is not possible even when many multidomain paths are available. However, as soon as the impact of inter-domain link delay increases, the error sharply drops. Indeed, we observe in Fig. 6-5b that a small relative error (< 5%) is achieved for $\rho$=30% with 160 multidomain paths, which is a reasonable configuration in realistic multidomain scenarios [Ch18]. This observation highlights the importance of accurate estimations of the inter-domain link delay components.

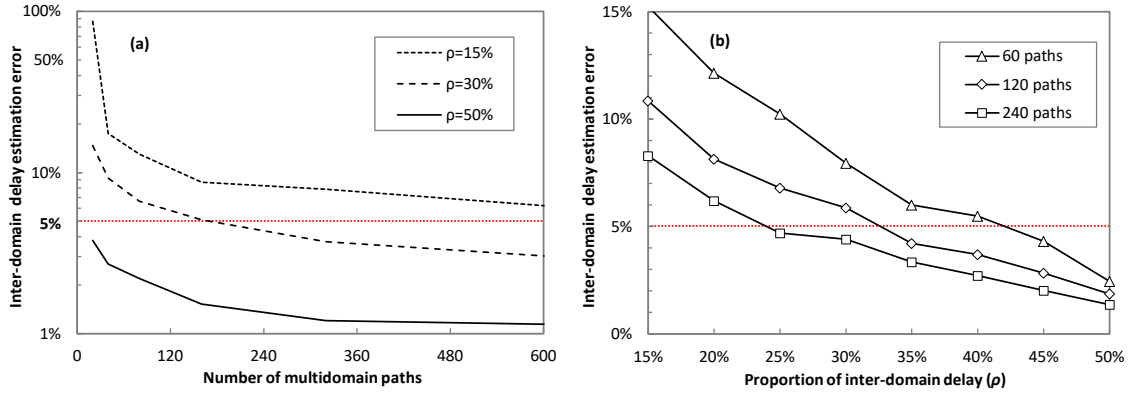


Fig. 6-5. Inter-domain link delay error estimation vs. number of multidomain paths (a) and proportion $\rho$ (b).

Assuming a realistic configuration of 240 paths and $\rho$=25%, let us now focus on analyzing the goodness of fit of the compound model in equation (6-1). We assume that block 6a in Fig. 6-4 is executed after one day of e2e traffic and delay measurements being available, i.e., the number of periods *Ttr* of monitoring data collection for model training purposes was set to 1,440 (1 day with 1 min. granularity).

These data were split 80-20% for training and validation, respectively and used to find the configuration of the DNN that returns the best performance in terms of accuracy with a moderated size to avoid overfitting. The backpropagation training algorithm using batches of 64 samples was considered; training took 30 min, which is much less that the time needed to collect used monitoring data. After evaluation of a wide range of number of hidden layers and size, a fully-connected DNN with three hidden layers and 10 neurons per layer implementing the hyperbolic tangent activation function was selected. Fig. 6-6 shows both training and validation loss, computed as the mean absolute deviation between the predicted and the actual delays as a function of the number of training epochs, for inter-domain links between domains 1 and 2. We observe that the convergence to an accurate and robust model is achieved after only a small number of training epochs; a common behavior observed in the rest of the inter-domain link models. The embedded table in Fig. 6-6 summarizes the prediction accuracy for the selected inter-domain link, where accuracy above 0.9 allows validating the reliability and high accuracy of the proposed method to estimate and model inter-domain link delay.

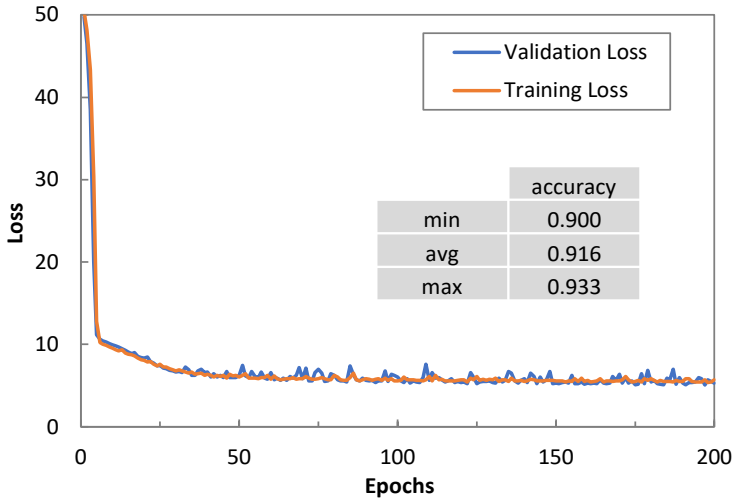


Fig. 6-6. Domain 1 to domain 2 link modeling performance.

Let us study now how increasing the monitoring period impacts the accuracy of the model. Fig. 6-7 presents the incremental maximum and the average error when larger monitoring periods are considered. We observe that the 1 min monitoring period can be relaxed to 5 or even 15 min without a major impact on the models' accuracy.

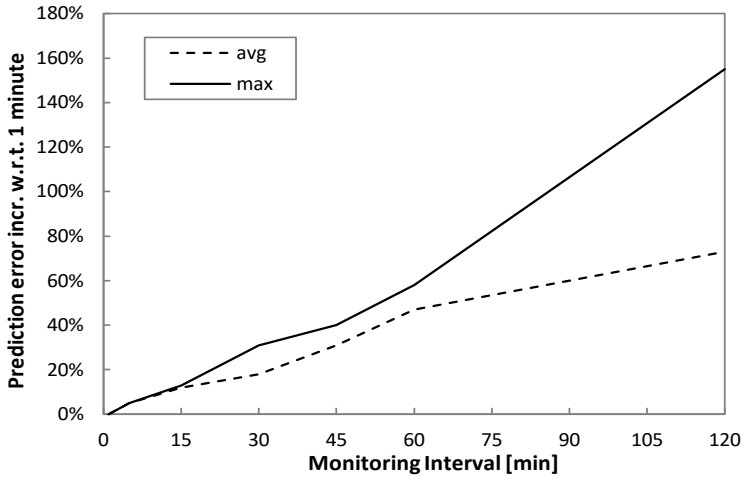


Fig. 6-7. Increment in prediction error vs. monitoring interval.

### 6.4.3 Benchmarking

Once the accuracy of the compound model has been validated, let us now compare this approach against two different methods that can also be used to compute and predict not only the e2e delay, but also the delay introduced by the domains and inter-domain links that a given path traverses. The methods are based on network tomography [He21] and INT [Ta21].

Since no information about the internal topology of the domains is available in the defined multi-operator scenario, we have applied network tomography to infer the delay of domains and inter-domain links from e2e measurements, predictions, and inter-domain routing. Specifically, the delay introduced by every component is computed so to minimize the mean square error between the approximated delay of every path, computed as the sum of the inferred delays of crossed segments, and the real e2e measurement. Regarding INT, the broker would be able to collect per-packet telemetry data, which includes the real delay introduced at every hop from source to destination. To hide internal domain topology, we assume that only edge and border routers add INT measurements to the packets.

For the sake of a fair comparison, we assume that all the models are obtained using the measurements obtained for path under study. Note that in the case of network tomography and INT, this assumption entails that there are not measurements available just after the path is established, whereas in the case of the compound model, domain models are available as measurements are collected during the commissioning testing. The same DNN configuration as for the compound model approach is used for these two approaches. We assumed the aforementioned realistic configuration (240 paths, $\rho$=25%) and conducted exhaustive evaluation of all the approaches for different values of parameter *Ttr*.

Let us first focus on the difference of the e2e delay prediction accuracy among the approaches for just one single path. Fig. 6-8 shows the measured and predicted e2e delays as a function of normalized input traffic x for the considered approaches. Fig. 6-8a plots the prediction before collecting any monitoring sample, i.e., just using the available information regarding the path distance in the case of tomography and INT -based approaches, and just with the intra-domain models in the case of the compound model.

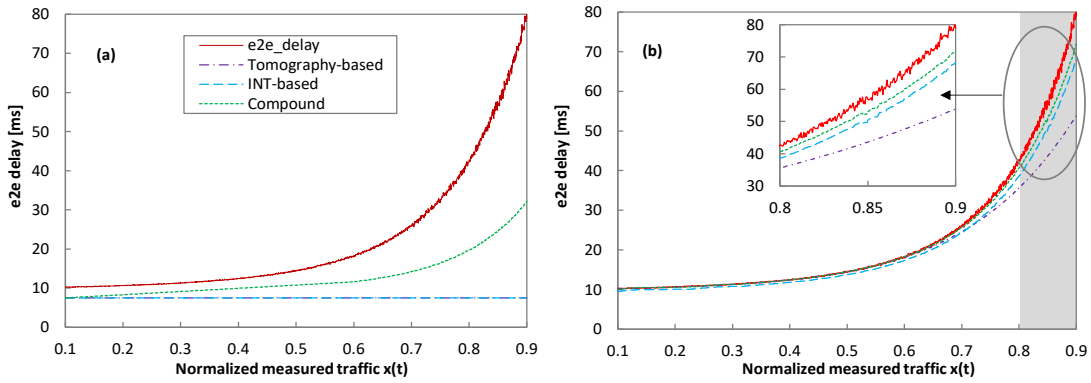


Fig. 6-8. End-to-end delay prediction example before (a) and after (b) training (*Ttr*=1440).

It is worth noting that the compound model approach offers the possibility to estimate a lower bound of the e2e delay consisting in the sum of all intra-domain delay components. In contrast, that bound, under the other approaches, is simply

reduced to the less informative transmission delay. Fig. 6-8b plots the prediction after data collection and model training for *Ttr*=1440 (i.e., 1 day with 1 min. granularity). In this case, assuming that a wide range of loads was observed during that period, all approaches quickly converge to the measured delay. The compound model and the INT-based approaches closely fit the perfect relation between load and e2e delay, whereas the tomography-based approach still needs some additional monitoring data to better learn the behavior of the e2e delay at high loads.

Fig. 6-9a shows the relative e2e prediction error (average and max for all 240 paths) as a function of *Ttr*, normalized to the value, where all approaches reached negligible average error (~1%). We observe that the compound model converges faster than other approaches, especially for the maximum error. Assuming that *Ttr* is chosen to guarantee a maximum error under a given target, Fig. 6-9b shows the relative anticipation of the compound model to achieve such target error w.r.t. the time needed under tomography or INT -based approaches (both since they exhibit same maximum error performance in Fig. 6-9a). Given the results, we can conclude that the compound approach reaches reasonable maximum error (around 10-15%) with a monitoring period between 20% and 35% shorter for training purposes.

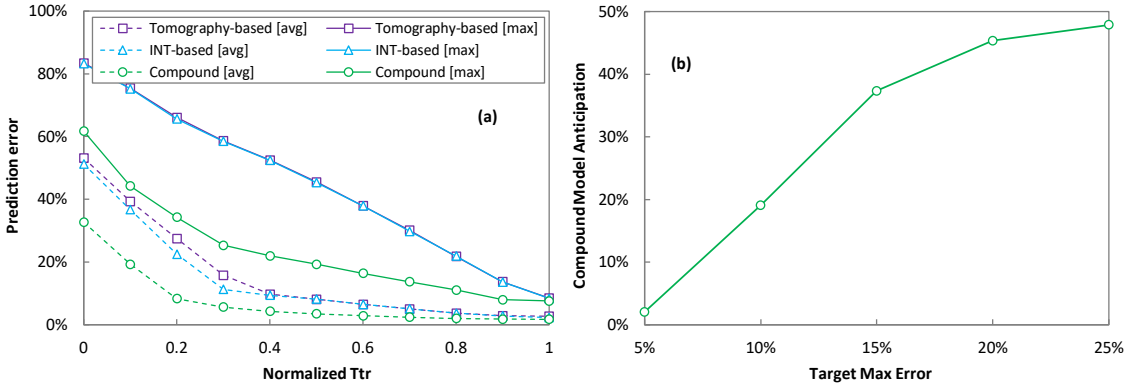


Fig. 6-9. End-to-end models' prediction error (a) and anticipation of compound model w.r.t benchmarking approaches (b).

Finally, delay prediction accuracy is evaluated in Fig. 6-10 for all the considered approaches as a function of *Ttr*. The tomography-based fails to produce accurate domain and inter-domain link delay models, as it can be observed in Fig. 6-10a and Fig. 6-10b, respectively. The rationale is related to the fact that paths with the same edge/border nodes can follow different inter-domain routes. The compound modeling approach exhibits the best combined performance; on the one hand, domain delay prediction error is constant and remarkably low since accurate models are available from the very beginning of operation, whereas inter-domain link delay prediction accuracy converges rapidly with time, following noticeable close to that given by the models trained with data from INT. In that regard, the INT-based approach produces very accurate models, but needs time to collect the required training dataset.

In conclusion, these results allow to validate the compound approach as a fast and accurate way to obtain e2e delay models and their per-segment components.

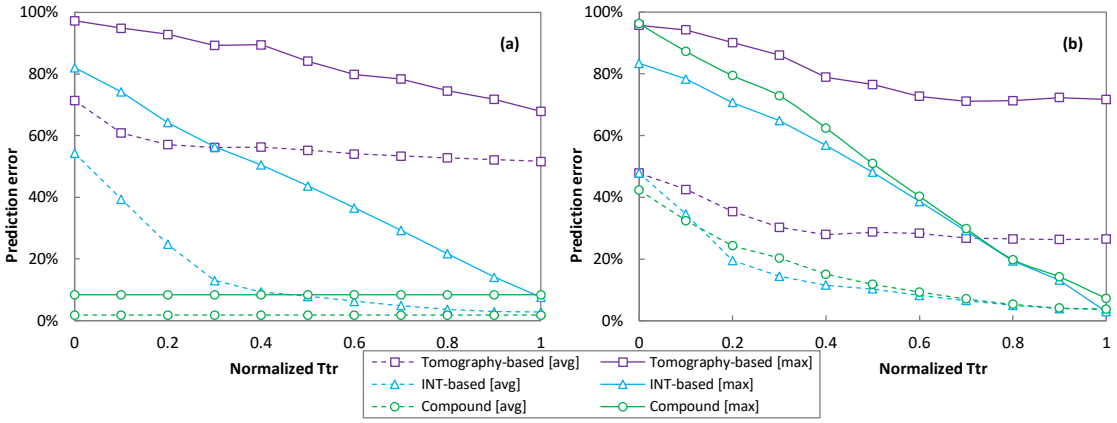


Fig. 6-10. Domain models' prediction error (a) and inter-domain link models' prediction error (b).

## 6.4.4 Intra-domain model correction

Let us now focus on validating the models and methods proposed in Section 6.3.2 to detect and localize inaccuracies in intra-domain models during the compound model training phase. Recall that inaccuracy detection is based on solving *Link Delay Disaggregation Bias* optimization problem and computing score $s$ in equation (6-11). As score $s$ is expected to increase when inaccuracies become larger, to demonstrate the validity of the detection and localization method we need to demonstrate that it is possible to set up a threshold value that discriminates inaccuracies with high precision, thus ensuring that accurate models robustly produce score values clearly under the threshold. Fig. 6-11 shows the score in the absence of inaccuracies for all the configurations of the number of paths and ρ already tested in the previous section. We observe that despite some oscillations in the score, the obtained results do not exceed $s$=1, whose value can be set as the threshold that separates accurate from inaccurate intra-domain models.

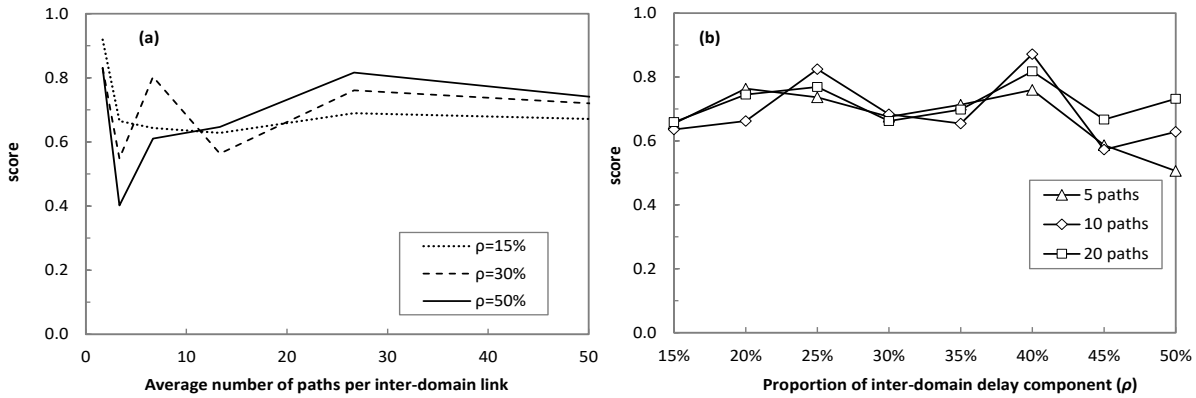


Fig. 6-11. Score values vs number of paths (a) and proportion ρ (b) in the absence of domain model inaccuracies.

Let us now compute the score s in the case of inaccurate intra-domain models. To this aim, we consider again the realistic scenario with 240 paths and $\rho$=25% and we have synthetically generated inaccuracies by adding some additional delay to the intra-domain prediction, thus emulating a hidden delay not considered during the intra-domain model training (See Chapter 5); inaccuracies range from 1 to 50 ms and were introduced at one single domain at a time. Fig. 6-12a shows the score as a function of the inaccuracy magnitude; curves for the minimum and maximum scores obtained for a given path and domain are presented. In the figures, we observe that the threshold defined as $s$=1 was clearly exceeded when magnitudes over 6-7 ms were introduced. Aiming at providing deeper insight into the performance of accuracy detection, Fig. 6-12b shows the precision computed as the probability of detecting a true inaccuracy, as a function of the inaccuracy magnitude for the range of magnitudes between 1 and 10 ms. Here, we can confirm 100% of precision for inaccuracy magnitude greater than 7 ms.

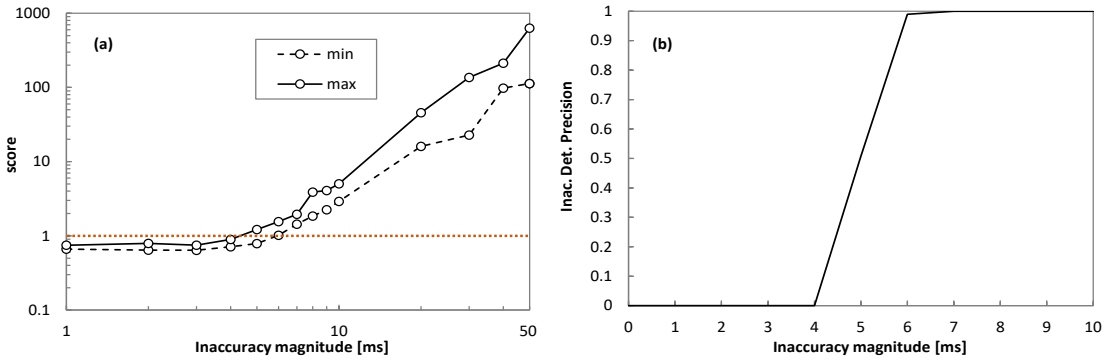


Fig. 6-12. Inaccuracy detection: score vs inaccuracy magnitude (a) and detection precision vs inaccuracy magnitude (b).

### 6.4.5 Inaccuracy localization

Once an inaccuracy has been detected, we need to localize it. Therefore, let us now focus on studying the performance of the inaccuracy localization procedure defined in Section6.3.3. Recall that we proposed the *Inaccuracy Localization* optimization problem to that aim, and the domain with the lowest bias $\beta_p$ is selected as the affected domain.

Fig. 6-13a shows the results obtained for the scenario used for the detection study, where an inaccuracy of 10 ms was introduced in domain 1; the bias (normalized to the domain that produces the maximum value) for each of the domains is shown. Note that domain 1 clearly presents the lowest bias (around 60% lower than the rest of the domains); the gap between inaccurate and accurate domains is represented by the double arrow in Fig. 6-13a. Fig. 6-13b shows the lowest bias (inaccurate domain) and the lowest bias among all accurate domains as a function of inaccuracy magnitude. We observe that the gap is large for all the inaccuracy magnitudes analyzed (from 10-50 ms), which supports a 100% localization precision in the studied range as the bias of accurate domains never drops below the bias of the inaccurate one.

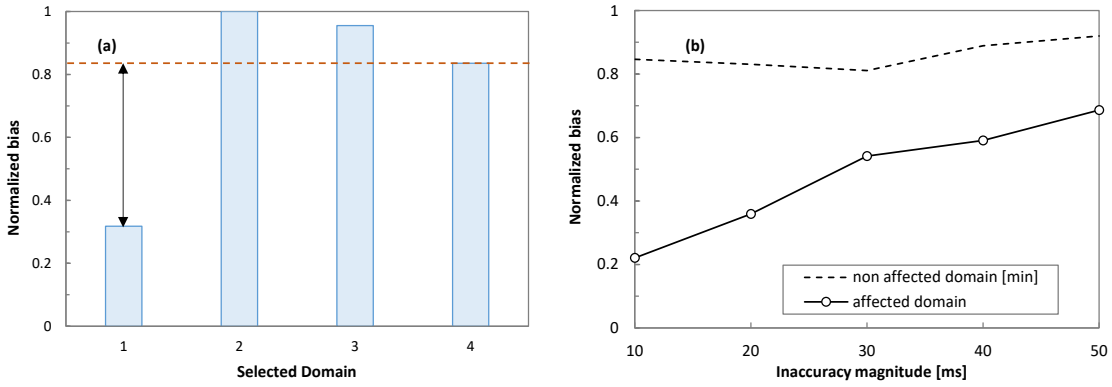


Fig. 6-13. Inaccuracy localization: example of 10 ms inaccuracy in domain 1 (a) and average results (b).

### 6.4.6 Using compound modeling to detect and localize inaccuracies in-operation

The previous studies focused on the training phase; we complement those results with a study of the use of the compound model once in-operation to detect inaccuracies and localize its potential root cause phase.

Let us first focus on analyzing how the compound model can be efficiently used if a sudden event in a domain, e.g., an internal domain reconfiguration, affects the accuracy of the intra-domain delay models for all the multidomain paths crossing the domain. We assume the previous network scenario with 240 paths and consider

that the proposed training procedure resulted in accurate compound models; the operation was emulated by generating 60 days of monitoring data samples.

In this case, the inaccuracy is detected by simply comparing the predicted and the measured e2e delays for every multidomain path and analyzing the resulting deviation; a threshold can be configured so its violation triggers its analysis. Fig. 6-14 shows the average deviation observed for a path crossing the inaccurate domain and for a path routed through other different (and accurate) domains, for different values of inaccuracy magnitude. Thus, setting up a deviation threshold around 15 ms allows us to detect significant inaccuracies above 10 ms. The localization of the inaccurate domain can be implemented by finding the common ones crossed by all affected paths (see [Bar21] for an example applied to single domain networks).

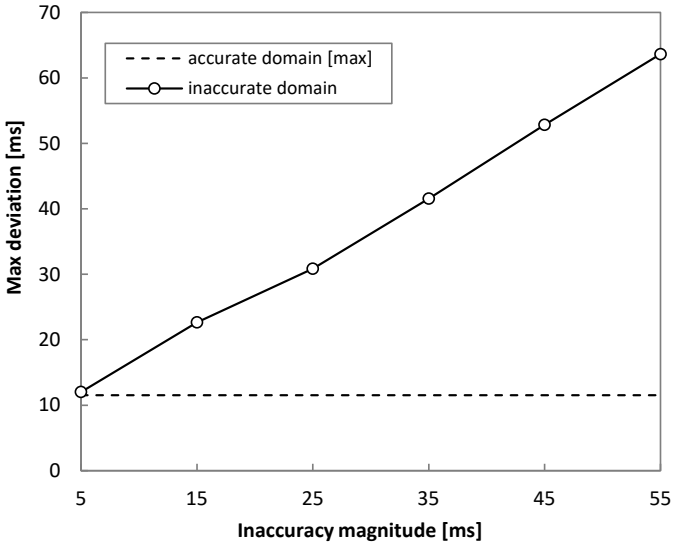


Fig. 6-14. Inaccuracy detection and localization for sudden inaccuracies.

Let us assume now a more realistic scenario where not all the paths in a domain are affected by an inaccuracy, its magnitude gradually increases with time, and it affects each path differently. Two different scenarios have been analyzed for a set of affected paths $P_{inac}$: *i) intra-domain increase*, where $|P_{inac}|$ is large; *ii) e2e increase*, where $|P_{inac}|$ is a small set and all paths share the same e2e route. To illustrate the difference between both scenarios, Fig. 6-15 shows the evolution of the number of inaccurate paths detected using the deviation analysis presented in Fig. 6-14 as a function of the time normalized to the instant when all the paths affected by the inaccuracy are detected. Once a path is detected, every component (domain or inter-domain link) is evaluated as a potential source of inaccuracy. To this aim, a list of *a priori* conditional assumptions for each component, which need to be previously defined, need to be evaluated. In this work, we simplify this list by considering only a maximum delay bound that cannot be exceeded in every domain and inter-domain link. Then, the potential set of inaccuracy sources is updated as soon as a path violates the delay bound until the inaccurate one is isolated.

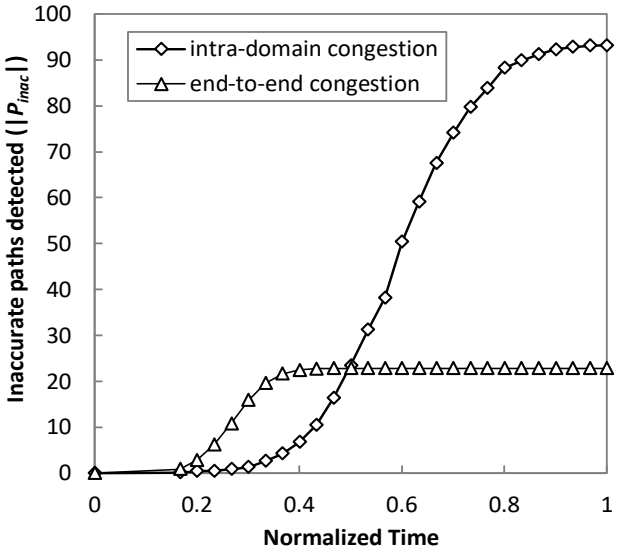


Fig. 6-15. Inaccuracy detection.

Fig. 6-16 shows the evolution of the localization accuracy in finding the inaccurate model component for the two scenarios considered. For comparison purposes, we have also considered the tomography-based as single model approach where, due to the lack of intermediate accurate model components, localization is done by choosing the domain or inter-domain link supporting the maximum number of inaccurate paths. We observe that the compound model allows almost perfect localization of the inaccurate component regardless of the scenario.

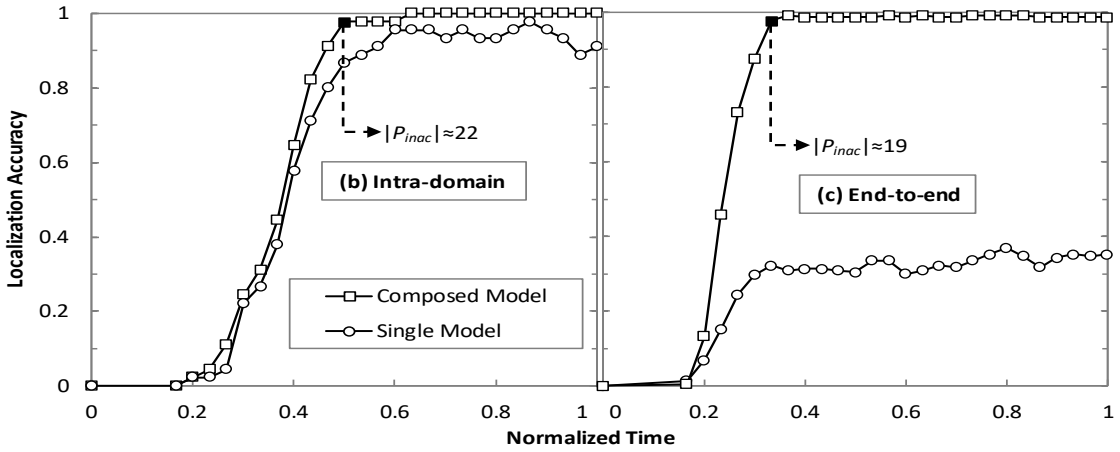


Fig. 6-16. Localization for gradual inaccuracies (a-b).

Note that the accuracy of the tomography-based approach is very dependent on the scenario, which discourages from utilizing it for e2e model evaluation purposes. Another result is that the minimum number of paths required to achieve virtually perfect localization accuracy (>99%) remains almost constant in the compound model approach, which is an important outcome as it allows identifying *a priori* condition (number of inaccurate paths detected) that can be applied to decide whether the

inaccurate component localization procedure is trustworthy or not and eventually validates the proposed approach based on a compound e2e modeling for different scenarios.

## 6.5 Concluding Remarks

This chapter proposed a coordination environment for multidomain networks, where domain networks and an inter-domain orchestrator (broker) consistently work for accurate analysis and modeling of e2e delay of multidomain paths. The proposed environment fosters cooperation by distributing tasks between the domains (in charge of modeling intra-domain delay components) and the broker (responsible for modeling inter-domain delay components). As a result of this cooperation, *compound e2e delay models* consisting of the sum of intra- and inter-domain components are obtained and used for multiple purposes, like QoS estimation for connectivity provisioning and reconfiguration upon anticipated QoS degradation.

A numerical evaluation of the proposed compound e2e delay modeling was conducted and compared against reference approaches, where models were trained by using e2e delay monitoring data only (tomography-based) or per-segment monitoring data (INT-based). The results show that: *i)* the inter-domain link delay can be accurately estimated by combining e2e monitoring data and intra-domain model predictions; *ii)* the broker can detect intra-domain model inaccuracies even when their magnitudes are small with respect to the total e2e delay; *iii)* the compound modeling approach converges to highly accurate e2e delay models faster than reference approaches; *iv)* once trained, the compound models can be effectively used to detect sudden in-operation inaccuracies under different potential scenarios, improving the performance of reference e2e delay models. These results validate the proposed cooperative e2e delay modeling architecture and methodology.

# Chapter 7

# KM in Intent-Based Networking scenarios

The Intent-Based Networking (IBN) paradigm targets at defining high-level abstractions, so network operators can define what are their desired outcomes without specifying how they would be achieved. The latter can be achieved by leveraging network programmability, monitoring and data analytics, as well as the key assurance component. In this last Chapter, we focus on applying KM in IBN scenarios, where cooperative intent operation is presented. Illustrative examples of intent-based operation and numerical results are presented and the obtained performance is discussed.

## 7.1 Cooperation among intents

### 7.1.1 Proactive Self-configuration

Autonomous network operation can be *reactive* (i.e., in response to events) or *proactive* (i.e., acting ahead of time). Let us illustrate the difference with an example, where a packet connection (*PkC*) is established and conveys a traffic flow with unknown traffic characteristics. Our target here is to allocate just enough capacity to ensure the required performance, which would optimize resource utilization. However, every different PkC supports services with different operational goals in terms of delay and throughput (e.g., keeping the total delay below a given maximum, or minimizing the capacity while ensuring zero packet losses, etc.), and so, the tailored capacity dimensioning is required.

Imagine that a policy-based management based on a fixed threshold (e.g., defined in terms of the ratio traffic volume over capacity) is set to operate the capacity of a PkC. Note that such operation can be highly reliable and it is based on a specific rule that is easily understood by human operators. However, deciding the value of the threshold requires knowledge of the traffic: *i*) a high threshold value (e.g., 90%) would result into poor performance coming from high delay, and it can be worse when the variability of the traffic is high; and *ii*) a low threshold value (e.g., 60%) would result into poor resource utilization. Therefore, some traffic analysis would be required. Further, since traffic characteristics can change over time, such analysis need to be continuously performed to change the operating model, when needed.

PkCs can be routed on top of virtual networks, where virtual links (*vLink*) are supported by the optical layer. Let us illustrate this problem with an example. Fig. 7-1 shows two PkCs (DC1-DC4 and DC2-DC3) that are established on top of a virtual network. Packet nodes are connected through vLinks, each supported by lightpaths on the optical layer. To minimize overprovisioning, such capacity is dynamically adjusted, thus enabling the dynamic vLink capacity management, e.g., by establishing and releasing parallel lightpaths between the end packet nodes.

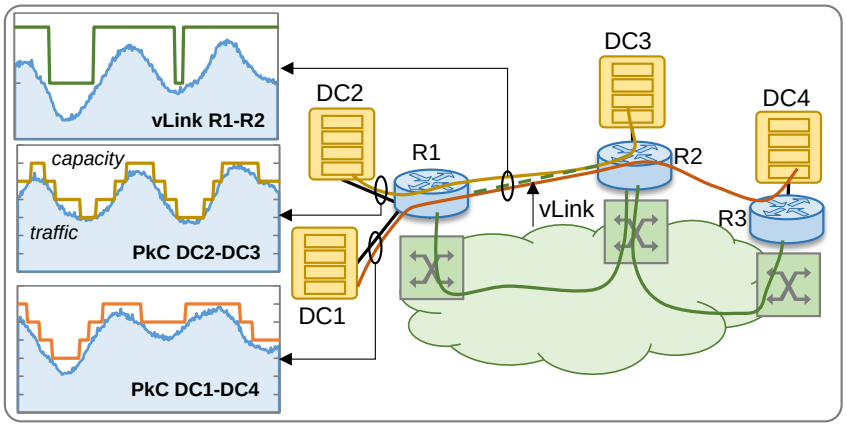


Fig. 7-1. Capacity operation of PkCs and vLinks.

Note that modifying the capacity of a PkC entails programming some rules in packet nodes and new capacity becomes immediately available. In contrast, adding more capacity to the vLink entails establishing a new lightpath, which requires some time (e.g., one minute). Therefore, vLink intents must make decisions with enough time to guarantee capacity availability. Such time depends, among others, of the packet traffic variation and thus, the value of the configured threshold could result into high delay and packet loss.

The inner graph for PkC DC2-DC3 in Fig. 7-1 shows the capacity adjustments performed assuming that the operational goal of the PkC is to minimize the allocated capacity to reduce connectivity costs, by following as close as possible the input traffic, while avoiding traffic loss.

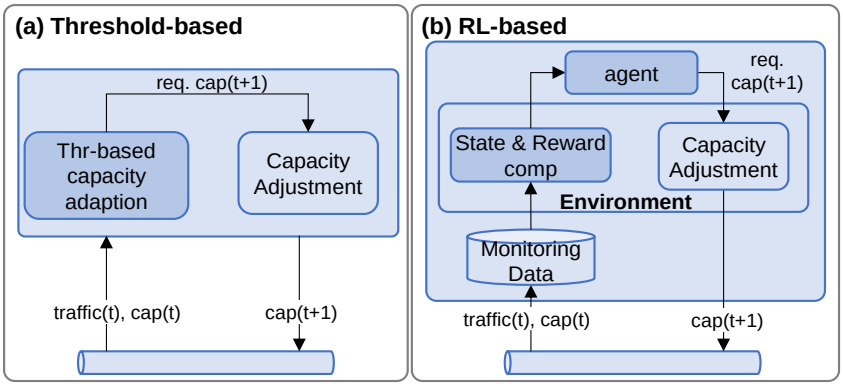


Fig. 7-2. Intent agents for PkCs and vLinks.

Fig. 7-2 presents two alternative approaches to operate the capacity of the connections (PkCs and vLinks), based on a simple threshold rule or based on an intelligent ML-based algorithm, in this case, RL. Every connection intent agent collects the amount of input traffic that is injected to the connection, as well as some other measurements, like packet loss and delay, and it determines the capacity of the connection that will be needed to meet the given operational goals for the next period (e.g., one minute). Such capacity can be used to program some rules in the packet nodes not only to increment the capacity but also, e.g., to adjust the amount of buffer at the input of the connection.

### 7.1.2 Cooperative Intent Operation and Transfer Knowledge

Although PkCs and related vLinks can work independently, making decisions based on the observed input traffic, some coordination might facilitate the overall operation. For instance, as a result of the capacity required by the PkCs, the capacity of the vLink needs to be reconfigured, as observed in Fig. 7-1. Nonetheless, if the available capacity of the vLink is exhausted, competition for the available capacity of the vLink would lead to poor performance for both PkCs.

A possible solution to avoid conflicts and countereffects between intent agents competing for common resources is to consider cooperation among them to ensure that they can achieve their operational goals. To illustrate such coordination, let us consider the multilayer scenario in Fig. 7-1. We assume that PkCs have different objectives. On the one hand, PkC DC1-DC4 requires that the maximum end-to-end delay is not violated, whereas PkC DC2-DC3 requires minimize overprovisioning. In spite of the subtle difference in the plots in Fig. 7-1 between both PkCs, the capacity of DC1-DC4 is always large enough with respect the input traffic to ensure that the delay added by the time spent in the queues is under the given maximum. Note that the capacity of DC2-DC3 is kept closer to the actual traffic. Considering the capacity requirements from PkCs, vLinks can be easily managed; the capacity of vLink R1-

R2 varies after adding or releasing one lightpath to adapt its aggregated capacity to the PkCs requirements, which motivates intent coordination.

To manage the capacity of the entities, the architecture in Fig. 7-3 supports a hierarchy of intents, where each intent agent is in line with that in Fig. 7-2. In the case of vLink intent agents, they receive as input the aggregated amount of input traffic in the vLink, its actual capacity, as well as the total capacity that PkCs will require for the next period, and are in charge of managing the vLinks capacity by establishing and tearing down lightpaths.

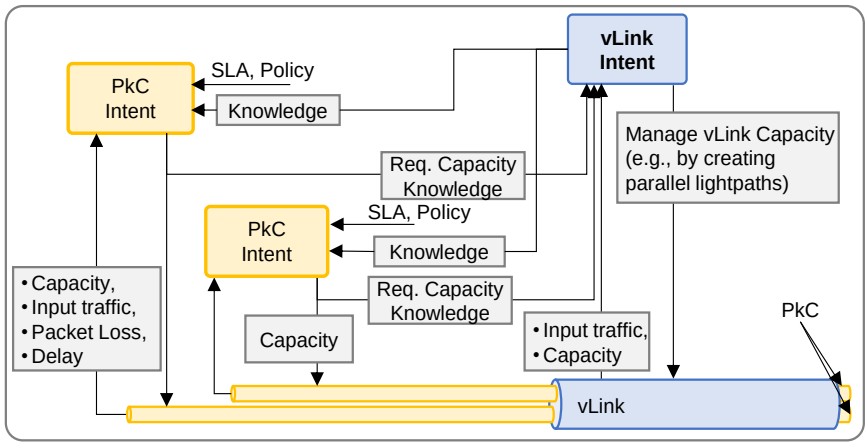


Fig. 7-3. Intent cooperation and transfer knowledge.

Besides, there is some knowledge that can be transferred from PkC intents to vLink intents, which cannot be anticipated by means of monitoring the (aggregated) traffic in the vLink. Knowledge that can be transferred include: *i*) traffic models for the PkC; *ii*) sudden capacity increase due to customer operational decisions (e.g., a pre-planned increase of productivity of a factory can lead to data traffic increase); or *iii*) PkC rerouting requiring new connectivity to be supported by the underlying network. This knowledge could be used by vLink intents to increase the capacity or, on the contrary, reject the request if no resources are available. Note that such rejection would be informed back to PkC intents, which will use that knowledge to reformulate their decisions and for finding alternatives to achieve the operational objectives.

## 7.2 Design of the Cooperative Intent Solution

In this section, we present the details of the design of intent cooperation for the problem presented in the previous section. The intents deployed for the individual PkCs take actions based on the traffic in the connection and cooperate with the vLink intent, which aggregates the capacity of the individual PkCs to decide the capacity

of the vLink. A stand-alone intent-based vLink capacity adaptation is extended to that end.

Fig. 7-4 shows the architecture for the hierarchical cooperation between PkC and vLink intents, where PkC intent agents implement a RL-based method, specifically *input traffic* $x(t)$ and *current vLink capacity* $z(t)$, that is collected periodically (e.g., every minute). Based on such analysis, the vLink intent agent determines the *target capacity* $z'(t+1)$ that should be allocated for the next period by using the learned optimal policy. The aggregation of the target capacity requested for every PkC is used as target capacity for the vLink. In addition, based on the enhanced RL-based operation scheme proposed in [OFC21], a LSTM-based traffic model can predict the evolution of the traffic and used to define the state for the RL agent.

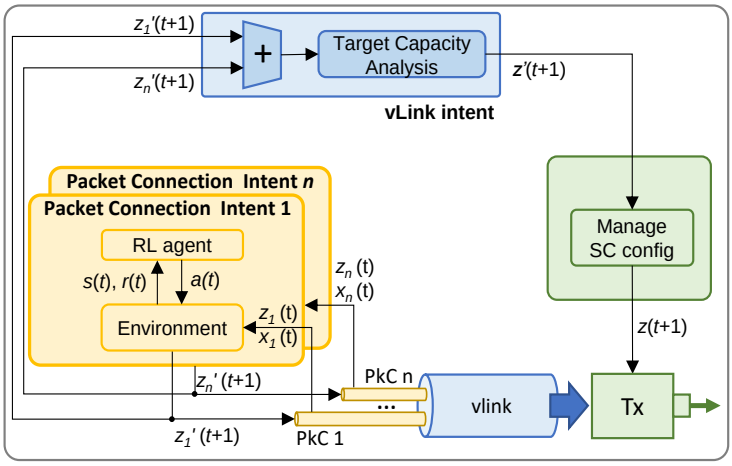


Fig. 7-4. Extended architecture with hierarchical intent cooperation.

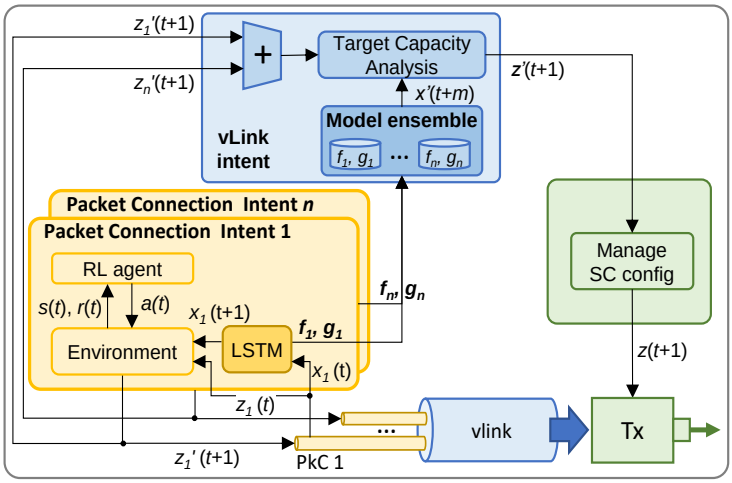


Fig. 7-5. Extended architecture with hierarchical intent cooperation and knowledge transfer.

Taking advantage of such prediction, we can transfer LSTM-based models from PkCs to vLink intents in order to enhance target capacity definition at the vLink (Fig. 7-5). In this approach, the vLink intent collects models from PkC intents and creates an ensemble that is used for long-term predictions $x'(t+m)$. The objective is to anticipate traffic variations and enhance the management of the underlying optical connection configuration, e.g., reducing the amount of SC changes to absorb both current and future traffic demand.

Specifically, a compound traffic model is proposed (see Fig. 7-6 for an illustrative example), with two components: *i*) an average profile component $f(t)$, that is a simple time-dependent function (e.g., polynomial or piece-wise linear) defined on a given periodicity, e.g., one day (Fig. 7-6a); and *ii*) a single step LSTM component that models the residual traffic $\xi(t)=x(t)-f(t)$ as a function of the last $w$ residual values ($g(t)$) (Fig. 7-6b). Note that $f$ models the overall periodic traffic evolution, while $g$ collects the specific variations observed in different periods, as well as other perturbation (peaks) that scape from the coarse granularity of $f$. Hence, the combination of both components provides high accurate predictions.

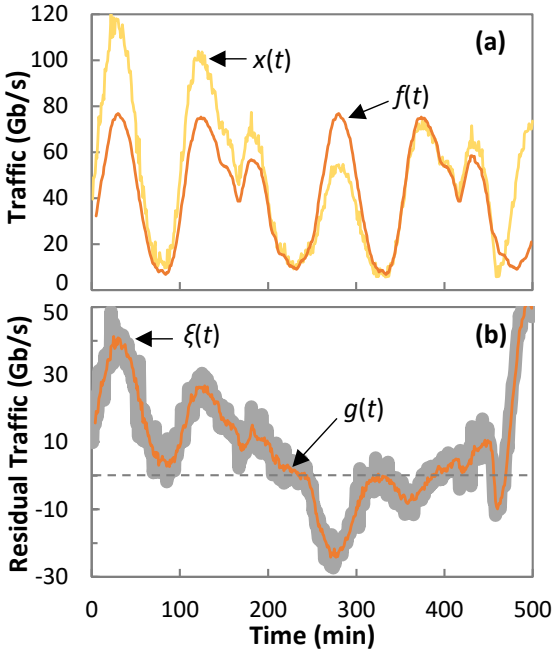


Fig. 7-6. Compound traffic model for PkCs.

The use of the proposed LSTM-based model for state definition is as follows: *i*) $x'(t+1)$ is obtained by estimating $\xi(t+1)$ with the LSTM model and adding it to $f(t+1)$. Then, $\Delta x(t)$ is computed as $\max(0, x'(t+1) - x(t))$ and used to obtain the load $l(t)$ (now redefined as $(x(t) + \Delta x(t)) / z(t)$), which is the main component of $s(t)$. This anticipation of traffic increase allows a better maximum delay assurance since it minimizes the risk of under-provisioning.

Any time a new prediction $x'(t+m)$ needs to be done, a multi-step procedure generates independent residual predictions from $\xi_i(t+1)$ to $\xi_i(t+m)$ with the $g_i$ models, using as input the residuals of the measured vLink traffic. Then, the average $\xi(t+m)$ is computed and added to $\sum_i f_i(t+m)$ to obtain the prediction $x'(t+m)$. The actual target capacity to be ensured is $\max(x'(t+m), z'(t+1))$.

## 7.3 Performance Evaluation

The numerical evaluation scenario detailed in Section 7.1 is used, where three PkCs A, B, C with maximum traffic 120, 60, and 60 Gb/s and different delay budgets have been considered. Both $f$ and $g$ models have been pre-trained after collecting 60 days of data. Periodicity of $f$ components was fixed to 1 day and $w$=120 minutes was selected for all $g$ components.

The accuracy of the proposed transfer knowledge scheme to predict vLink traffic is illustrated in Fig. 7-7a. The vLink traffic is compared against the prediction from model ensemble for $m$=60 min. Models learned by different PkCs intent agents for the short 1-min scope can be aggregated and used by the vLink for a much longer time scope with remarkable accuracy. Fig. 7-7b shows the delay at the source node for all PkCs, assuming that PkCs leave through 100/200 Gb/s interfaces. We observe that RL-based operation at the PkC level allows achieving the targeted differentiated delay performance.

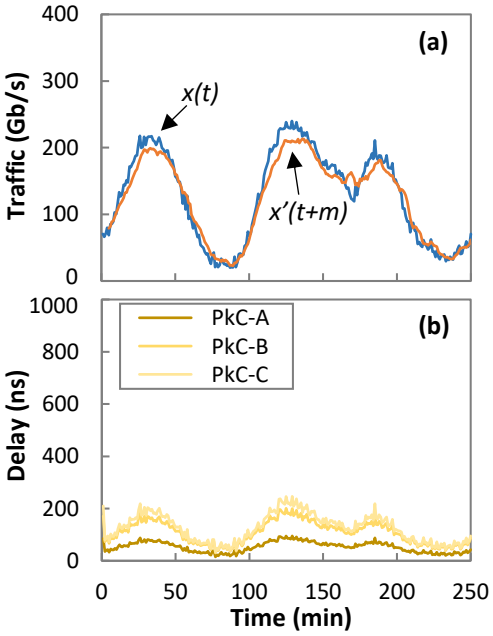


Fig. 7-7. PKC-vLink intent cooperation performance.

From Fig. 7-7, we observe that the requested target capacity for every PkC is accurate and well-fitted so, the sum of all target capacities to be considered by the

vLink intent agent results into an overall capacity that meets PkCs needs. Then, we conclude that knowledge transfer is potentially very useful for vLink intents as it enables prediction capabilities without the need of learning the models.

Finally, Fig. 7-8 and Table 7-1 compare hierarchical cooperation without and with knowledge transfer, named *hierarchical* (Fig. 7-4) and *transfer* (Fig. 7-5), respectively.

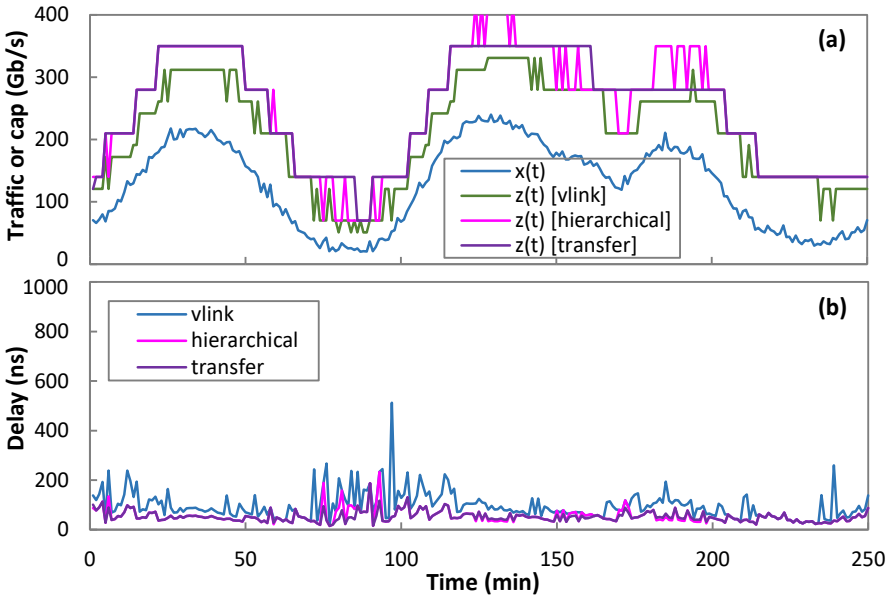


Fig. 7-8. Comparative results.

*Table 7-1. Cooperative IBN summary*

| Method | Over-Provisioning (Tb/day) | Num SC Changes | Delay (ns) | | |
|---|---|---|---|---|---|
| | | | min | avg | max |
| **vLink** | 20.8 | 70 | 21 | 96 | 512 |
| **Hierarchical** | 29.8 | 50 | 15 | 54 | 233 |
| **Transfer** | 29.8 | **16** | **15** | **52** | **187** |

For comparison purposes, RL-based vLink intent performance is considered. Results of the vLink allocated capacity and introduced delay are presented in Fig. 7-8a and Fig. 7-8b, respectively. In view of the graphs, the requirements from the PkCs to achieve differentiated delay performance result in a higher capacity that cannot be successfully learnt by the vLink intent. Interestingly, when PkC operation is intent-based, the number of SC changes at the optical layer reduces noticeably. This fact points out the benefits of hierarchical intent cooperation to simplify multilayer operation. This reduction is even larger when knowledge transfer is implemented; really few SC changes are enough to accommodate the same overall capacity in a more intelligent way. Moreover, the delay contribution introduced by the vLink is

greatly reduced. We can conclude that hierarchical intent cooperation with knowledge transfer is the option that provides the best trade-off between the achievement of the operational goals and resource utilization and management.

## 7.4 Concluding Remarks

In this last chapter we have focused on applying KM on IBN scenarios. IBN allows network operators to define *what* are their desired outcomes without specifying *how* such outcomes would be achieved. IBN can be fueled by the use of ML algorithms.

An illustrative example of multilayer networks has been used to showcase cooperative intent operation and transfer knowledge. We assumed that intent agents adjust the capacity of a vLink as a function of the input traffic and proposed a combined LSTM and RL approach for dynamic PkC capacity allocation. The LSTM models for every PkC can be shared to vLink intents to anticipate long-term traffic changes. Numerical results were presented and discussed.

# Chapter 8

# Closing Discussion

## 8.1 Main Contributions

This Ph. D. thesis focuses on applying KM to multi-layer and multi-domain networks and showcasing its relevance under the IBN paradigm. The main contributions are summarized as following:

- First, the KM process in the context of networking was introduced in Chapter 4 and based on four pillars: i) discovery, ii) share, iii) assimilating, and iv) using knowledge. The approach leads to benefits such as discovering and disseminating knowledge that can be used to adapt the network configuration to variable conditions without human intervention. To highlight the KM robustness to adapt to different application, we implemented alternative sharing strategies in purely centralized and purely distributed use cases. The data-based approach reduced convergence error to a negligible amount compared when no data is shared. In consequence, we evaluated the benefit of using knowledge extension and consolidation by means of model assimilation. Results showed similar convergence error, while reducing shared data volumes.
- Next, KM was implemented in a multilayer network scenario. In this regard, Chapter 5 was devoted to proposed and experimentally evaluated the PILOT methodology to discover knowledge. The knowledge is shared and utilized for predicting the performance of connectivity services as throughput and latency. Remarkable results showed the efficiency of using real measurements obtained from active probs to tune a traffic simulation. Additionally, training and validating ML models with such large realistic traffic dataset quickly converges and reduces error to a minimal value.

- Chapter 6 applied KM in a multi-domain network, where a cooperative environment is proposed to enable cooperation between a broker plane and network domains. Each domain discover knowledge in terms of intra domain model and share with the broker to estimate e2e delay. Numerical evaluation demonstrated the approach strength to detect and localize intra-domain inaccuracies. Besides, such compound models converge faster to an accurate e2e delay and target prediction error.
- Finally, in Chapter 7 we studied the application of KM application in an IBN multilayer scenario. In this regard, a compound LSTM-based model with discovered knowledge from different PkC intent agents is shared. Results exhibited noticeable performance for traffic prediction and resulted in lower delay, as well decreased number of reconfigurations.

## 8.2 List of Publications

### 8.2.1 Publications in Journals

[JOCN22] L. Velasco, S. Barzegar, **F. Tabatabaeimehr**, and M. Ruiz, "Intent-Based Networking for Optical Networks [Invited Tutorial]," IEEE/OSA Journal of Optical Communications and Networking, vol.14, pp. A11-A22, 2022.

[TNSM21] **F. Tabatabaeimehr**, M. Ruiz, C.-Y. Liu, X. Chen, R. Proietti, S. J. B. Yoo, and L. Velasco, "Cooperative Learning for Disaggregated Delay Modeling in MultiDomain Networks," IEEE Transactions on Network and Service Management, vol. 18, pp. 3633-3646, 2021.

[JLT20] M. Ruiz, M. Ruiz, **F. Tabatabaeimehr**, Ll. Gifre, S. López-Buedo, J. López de Vergara, O. González, and L. Velasco, "Modeling and Assessing Connectivity Services Performance in a Sandbox Domain," IEEE/OSA Journal of Lightwave Technology (JLT), vol. 38, pp. 3180-3189, 2020.

[JOCN20] M. Ruiz, **F. Tabatabaeimehr**, and L. Velasco, "Knowledge Management in Optical Networks: Architecture, Methods and Use Cases [Invited]," IEEE/OSA Journal of Optical Communications and Networking, vol. 12, pp. A70-A81, 2020.

### 8.2.2 Publications in Conferences

[OFC21] **F. Tabatabaeimehr**, S. Barzegar, M. Ruiz and L. Velasco, "Combining Long-Short Term Memory and Reinforcement Learning for Improved Autonomous Network Operation," in Proc. IEEE/OSA Optical Fiber Communication Conference (OFC), 2021.

[ECOC20] **F. Tabatabaeimehr**, M. Ruiz and L. Velasco, "Supporting Beyond 5G Applications by Coordinating AI-based Intent Operation. An Example for Multilayer Metro Networks," in Proc. European Conference on Optical Communication (ECOC), 2020.

[ICTON20] Luis Velasco, **Fatemehsadat Tabatabaeimehr**, and Marc Ruiz, "Knowledge Management in Optical Networks," in Proc. IEEE International Conference on Transparent Optical Networks (ICTON), 2020.

[ICTON19] **F. Tabatabaeimehr**, M. Ruiz, and L. Velasco, "Distributed and centralized options for self-learning (Invited)," in Proc. IEEE International Conference on Transparent Optical Networks (ICTON), 2019.

## 8.3 List of Research Projects

### 8.3.1 European Funded Projects

- **METRO-HAUL:** METRO High bandwidth, 5G Application-aware optical network, with edge storage, compute and low Latency, H2020-ICT-2016-2. (G.A. 761727).
- **B5G-OPEN: Beyond 5G – Optical Network Continuum**, H2020-ICT-2020-2. (G.A. 101016663).

### 8.3.2 National Funded Projects

- **IBON:** AI-Powered Intent-Based Packet and Optical Transport Networks and Edge and Cloud Computing for Beyond 5G, AEI PID2020-114135RB-I00, 2021-2024.
- **TWINS:** cogniTive 5G application-aware optical metro netWorks Integrating moNitoring, data analyticS and optimization, MINECO TEC2017-90097-R, 2018-2020.

### 8.3.3 Pre-doctoral Scholarship

- Pre-doctoral scholarship "FI-DGR 2019" funded by "Agència de Gestió d'Ajuts Universitaris i de Recerca" (AGAUR), Generalitat de Catalunya,2019-2022.

## 8.4 Collaborations

I did a three-month research stay at Inria Sophia Antipolis-Méditerranée, and I collaborated with the "Combinatorics, Optimization, and Algorithms for Telecommunications" (COATI) team. Our research carried out is related to the application of KM in IBN scenarios (G.4).

I had the opportunity to collaborate with University of California, Davis (UCDavis) to propose the cooperative environment, where the broker system compounds models to provide 2e connectivity services (G.3).

In addition, I was involved in a fruitfully collaboration with teams from Telefónica, Nokia Bell Labs France, Universidad Autónoma de Madrid and Naudit HPCN to assess the PILOT methodology (G.2).

## 8.5 Topics for Further Research

In Chapter 7 we examined the robustness of LSTM for short-term sharp traffic changes while keep the long-term view. Although overall accuracy of the LSTM-based predictor is noticeable, such a model was trained for a specific flow and it will be not accurate when used to model other flows, i.e., an individual model is required for each traffic flow. In this regard, we are already working towards a LSTM-based model enabling multiple traffic flows for dynamic connection capacity allocation. In this regard, we are exploring methods to update LSTM-based model in online operation. The new approach will work as a pure continual learning, while avoiding unnecessary re-training.

# List of Acronyms

| | |
|---|---|
| AI | Artificial intelligence |
| ANN | Artificial Neural Networks |
| ATA | Autonomic Transmission Agent |
| BERT | Bit Error Rate Test |
| BS | Burst Size |
| CO | Central Offices |
| COM | Control, Orchestration, and Management |
| DNN | Deep Neural Networks |
| DT | Decision Trees |
| e2e | end-to-end |
| FEC | Forward Error Correction |
| FIFO | First-In-First-Out |
| GPS | Global Positioning System |
| GUI | Graphical User Interface |
| IBN | Intent-Based Networking |
| IBR | Inter-arrival Burst Rate |
| INT | In-band Network Telemetry |
| IPPM | IP Performance Measurement |
| KM | Knowledge Management |
| KPI | key Performance Indicators |
| LSTM | Long Short-Term Memory |
| MDA | Monitoring and Data Analytics |
| ML | Machine learning |

| | |
|---|---|
| Multi-AS | Multi-operator |
| NBI | REST-API-based northbound interface |
| NFV | Network Function Virtualization |
| NFVI | Network Function Virtualization Infrastructure |
| NFVO | Network Function Virtualization Orchestration |
| NS | Network Service |
| OD | Origin-Destination |
| OSA | Optical Spectrum Analyzers |
| p2mp | point-to-multipoint |
| p2p | point-to-point |
| PRBS | Pseudorandom Binary Sequence |
| PS | Packet Size |
| QoS | Quality of Service |
| SDN | Software-Defined Networking |
| SOP | State of Polarization |
| SVM | Support Vector Machines |
| TMN | Telecommunications Management Network |
| VNT | Virtual Network Topology |

# References


[ACTN] D. Ceccarelli and Y. Lee, editors, “Framework for Abstraction and Control of TE Networks (ACTN)”, IETF RFC8453, 2018.

[APV17] A. P. Vela, M. Ruiz and L.Velasco, “Distributing Data Analytics for Efficient Multiple Traffic Anomalies Detection,” Elsevier Computer Comm., vol. 107, pp. 1-12, 2017.

[Ba15] A. Bartolucci, K. Singh, and S. Bae, *Introduction to Statistical Analysis of Laboratory Data*, Wiley, 2015.

[Bar21] S. Barzegar, M. Ruiz, A. Sgambelluri, F. Cugini, A. Napoli, and L. Velasco, "Soft-Failure Detection, Localization, Identification, and Severity Prediction by Estimating QoT Model Input Parameters," IEEE Transactions on Network and Service Management, vol. 18, pp. 2627-2640, 2021.

[Be20] A. Bernal, M. Richart, M. Ruiz, A. Castro, and L. Velasco, “Near Real-Time Estimation of End-to-End Performance in Converged Fixed-Mobile Networks,” Elsevier Computer Comm., vol. 150, pp. 393-404, 2020.

[Bi21] J. Bi, X. Zhang, H. Yuan, J. Zhang and M. Zhou, “A Hybrid Prediction Method for Realistic Network Traffic with Temporal Convolutional Network and LSTM,” in IEEE Transactions on Automation Science and Engineering, 2021.

[Ca02] G. Casella and R. Berger, “Statistical Inference,” 2nd ed., Duxbury/Thomson Learning, 2002.

[Ca16] A. Castro, L. Velasco, L. Gifre, C. Chen, J. Yin, Z. Zhu, R. Proietti, and S. B. J. Yoo, “Brokered Orchestration for End-to-End Service Provisioning across Heterogeneous Multi-Operator (Multi-AS) Optical Networks,” IEEE/OSA J. of Lightwave Tech., vol. 34, pp. 5391-5400, 2016.

[Ch18] X. Chen, R. Proietti, H. Lu, A. Castro, and S. J. B. Yoo,“Knowledge-Based Autonomous Service Provisioning in Multi-Domain Elastic Optical Networks,” IEEE Comm. Mag., vol. 56, pp. 152-158, 2018.

[Ch19.1] X. Chen, B. Li, R. Proietti, Z. Zhu, and S. J. B. Yoo ,“Multi-Agent Deep Reinforcement Learning in Cognitive Inter-Domain Networking with Multi-Broker Orchestration,” in Proc. OFC, 2019.

[Ch19.2] X. Chen, B. Li, R. Proietti, C.-Y. Liu, Z. Zhu, and S. J. B. Yoo,"Demonstration of distributed collaborative learning with end-to-end QoT estimation in multi-domain elastic optical networks," OSA Optics Express vol. 27, pp. 35700-35709, 2019.

[Ch83] V. Chvatal, Linear Programming, Ed. Freeman, 1983.

[Cha10] O. Chapelle, B. Schlkopf, and A. Zien, Semi-Supervised Learning, 1st ed. The MIT Press, 2010.

[Di00] T.G. Dietterich, "Ensemble Methods in Machine Learning," in Proc. Multiple Classifier Systems (MSC), Lecture Notes in Computer Science, vol 1857, 2000.

[Do20] R. Dong, Ch. She, W. Hardjawana, Y. Li and B. Vucetic, "Deep Learning for Radio Resource Allocation with Diverse Quality-of-Service Requirements in 5G," IEEE Trans. Wireless Comm., 2020.

[Ea11] S. Eakambaram and R. Elangovan, Least Absolute Deviation Regression Theory and Methods: Monograph on LAD Regression Theory, LAP Lambert, 2011.

[Fi19] S. Fichera, R. Martínez, B. Martini, M. Gharbaoui, R. Casellas, D R. Vilalta, R. Muñoz, and P. Castoldi, "Latency-Aware Resource Orchestration in SDN-Based Packet Over Optical Flexi-Grid Transport Networks," IEEE/OSA J. of Optical Communications and Networking, vol. 11, pp. B83-B96, 2019.

[Fin19] N. Finn, J-Y. L. Boudec, E. Mohammadpour, J. Zhang, B. Varga, and J. Frank, "DetNet Bounded Latency," IETF draft work-in-progress, 2019.

[Gi18] Ll. Gifre, J.-L. Izquierdo-Zaragoza, M. Ruiz, and L. Velasco, "Autonomic Disaggregated Multilayer Networking," IEEE/OSA Journal of Optical Communications and Networking, vol. 10, pp. 482-492, 2018.

[Gu19] V. Le Guen, N. Thome, "Shape and Time Distortion Loss for Training Deep Time Series Forecasting Models," NeurIPS, 2019.

[He21] T. He, L. Ma, A. Swami, and D. Towsley, *Network Tomography: Identifiability, Measurement Design, and Network State Inference*, Cambridge University Press, 2021.

[Kn00] G. Knott, Interpolating cubic splines, Birkhäuser, 2000.

[Kr11] D. P. Kroese, T. Taimre, and Z. Botev, Handbook of Monte Carlo Methods, Wiley, Wiley, 1st ed., 2011.

[Kr20] F. Krasniqi, J. Elias, J. Leguay, and A. E. C. Redondi, "End-to-end Delay Prediction Based on Traffic Matrix Sampling," in Proc. IEEE INFOCOM 2020.

[Le18] R. Leira, J. Aracil, J. E. López de Vergara, P. Roquero, and I. González, "High-speed optical networks latency measurements in the microsecond timescale with software-based traffic injection," Optical Switching and Networking, vol. 29, pp. 39-45, 2018.

[Lo19] J. López de Vergara, M. Ruiz, Ll. Gifre, M. Ruiz, L. Vaquero, J. Zazo, S. López-Buedo, O. González de Dios, and L. Velasco, "Demonstration of 100 Gbit/s Active Measurements in Dynamically Provisioned Optical Paths," in Proc. European Conference on Optical Communication, 2019.

[Ma02] D. Mandic and J. Chambers, Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability, Wiley, 2001.

[Ma11] S. Marsland, *Machine Learning: An Algorithmic Perspective*. CRC Press, 2011. [Online]. Available: https://books.google.de/books?id= n66O8a4SWGEC.

[Metro-Haul18] METRO-HAUL project, "Deliverable D3.1: Selection of metro node architectures and optical technologies," [on-line: https://metro-haul.eu], 2018.

[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," IEEE/OSA Journal of Optical Communications and Networking (JOCN), vol. 9, pp. 1202-1211, 2017.

[Mo17.2] F. Morales, M. Ruiz, Ll. Gifre, L. M. Contreras, V. Lopez, and L. Velasco, "Virtual Network Topology Adaptability based on Data Analytics for Traffic Prediction," IEEE/OSA J. of Optical Comm. and Networking, vol. 9, pp. A35-A45, 2017.

[MR16.1] M. Ruiz, J. Ramos, G. Sutter, S. L. Buedo, J.E. López de Vergara, and C. Sisterna, "Harnessing Programmable SoCs to Develop Cost-effective Network Quality Monitoring Devices," in Proc. FPL, 2016.

[MR16.2] M. Ruiz, J. Ramos, G. Sutter, J. E. L. Vergara, S. Lopez-Buedo, and J. Aracil, "Accurate and affordable packet-train testing systems for multi-Gb/s networks," IEEE Comm. Magazine, vol. 54, pp. 80-87, 2016.

[Ni20] S. Nihale, S. Sharma, L. Parashar, and U. Singh, "Network Traffic Prediction Using Long Short-Term Memory," in Proc. ICESC, 2020.

[Pa16] F. Paolucci, V. Uceda, A. Sgambelluri, F. Cugini, O. Gonzales de Dios, V. Lopez, L. M. Contreras, P. Monti, P. Iovanna, F. Ubaldi, T. Pepe, and P. Castoldi, "Interoperable Multi-Domain Delay-aware Provisioning using Segment Routing Monitoring and BGP-LS Advertisement," in Proc. European Conference on Optical Communication (ECOC), 2016.

[Ra18] D. Rafique and L. Velasco, "Machine Learning for Optical Network Automation: Overview, Architecture and Applications," IEEE/OSA J. of Optical Comm. and Networking, vol. 10, pp. D126-D143, 2018.

[Re97] A. C. Rencher, Multivariate Statistical Inference and Applications, Wiley, 1st ed., 1997.

[Rk20] K. Rusek, J. Suárez-Varela, P. Almasan; P. Barlet-Ros, and A. Cabellos-Aparicio, "RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN," IEEE J. on Sel. Areas in Comm., vol. 38, pp. 2260-2270, 2020.

[Ru15] M. Ruiz and L. Velasco, "Serving Multicast Requests on Single Layer and Multilayer Flexgrid Networks," IEEE/OSA J. of Optical Communications and Networking, vol. 7, pp. 146-155, 2015.

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

[Ru16.2] M. Ruiz, J. Ramos, G. Sutter, J. E. Lopez de Vergara, S. Lopez-Buedo, and J. Aracil, "Accurate and affordable packet-train testing systems for multi-Gb/s networks," IEEE Comm. Magazine, vol. 54, pp. 80-87, 2016.

[Ru18] M. Ruiz, F. Coltraro, and L. Velasco, "CURSA-SQ: A Methodology for Service-Centric Traffic Flow Analysis," IEEE/OSA J. of Optical Comm. and Networking, vol. 10, pp. 773-784, 2018.

[Ru19] M. Ruiz, F. Boitier, P. Layec, and L. Velasco, "Self-Learning Approaches for Real Optical Networks," in Proc. IEEE/OSA Optical Fiber Communication Conference (OFC), 2019.

[Sc01] B. Scholkopf and A. L. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press Cambridge, 2001.

[Se90] P. Senge, *The Fifth Discipline: The Art and Practice of the Learning Organization*, Doubleday/Currency, 1990.

[Sh19] B. Shariati, M. Ruiz, J. Comellas, and L. Velasco, "Learning from the Optical Spectrum: Failure Detection and Identification [Invited]," IEEE/OSA Journal of Lightwave Technology, vol. 37, pp. 433-440, 2019.

[So20] A. Solano and L. Contreras, "Information Exchange to Support Multi-Domain Slice Service Provision for 5G/NFV," in Proc. IFIP Networking, pp. 773-778, 2020.

[Ta21] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li,"In-band Network Telemetry: A Survey," Computer Networks, vol. 186, 2021.

[Tr18] H. D. Trinh, L. Giupponi and P. Dini, "Mobile Traffic Prediction from Raw Data Using LSTM Networks," IEEE Annual International Symposium on Personal, Indoor and Mobile Radio Communications *(PIMRC)*, 2018.

[Ve13] L. Velasco, P. Wright, A. Lord, and G. Junyent, "Saving CAPEX by Extending Flexgrid-based Core Optical Networks towards the Edges," IEEE/OSA Journal of Optical Communications and Networking, vol. 5, pp. A171-A183, 2013.

[Ve17.1] L. Velasco and M. Ruiz, Provisioning, Recovery and In-operation Planning in Elastic Optical Networks, Wiley, 1st ed., 2017.

[Ve17.2] L. Velasco, A. P. Vela, F. Morales, and M. Ruiz, "Designing, Operating and Re-Optimizing Elastic Optical Networks," IEEE/OSA J. of Lightwave Tech., vol. 35, pp. 513-526, 2017.

[Ve18.1] L. Velasco, A. Sgambelluri, R. Casellas, Ll. Gifre, J.-L. Izquierdo-Zaragoza, F. Fresi, F. Paolucci, R. Martínez, and E. Riccardi, "Building Autonomic Optical Whitebox-based Networks," IEEE/OSA J. of Lightwave Technology, vol. 36, pp. 3097-3104, 2018.

[Ve18.2] L. Velasco, Ll. Gifre, J.-L. Izquierdo-Zaragoza, F. Paolucci, A. P. Vela, A. Sgambelluri, M. Ruiz, and F. Cugini, "An Architecture to Support Autonomic Slice Networking [Invited]," IEEE/OSA J. of Lightwave Tech., vol. 36, pp. 135-141, 2018.

[Ve19.1] L. Velasco, B. Shariati, F. Boitier, P. Layec, and M. Ruiz, "A Learning Life-Cycle to Speed-up Autonomic Optical Transmission and Networking Adoption," IEEE/OSA Journal of Optical Communications and Networking, vol. 11, pp. 226-237, 2019.

[Ve19.2] L. Velasco, A. Chiadò Piat, O. González, A. Lord, A. Napoli, P. Layec, D. Rafique, A. D'Errico, D. King, M. Ruiz, F. Cugini, and R. Casellas, "Monitoring and Data Analytics for Optical Networking: Benefits, Architectures, and Use Cases," IEEE Network, vol. 33, pp. 100-108, 2019.

[Ve19.3] L. Velasco, R. Casellas, S. Llana, Ll. Gifre, R. Martinez, R. Vilalta, R. Muñoz, and M. Ruiz, "A Control and Management Architecture Supporting Autonomic NFV Services," Photonic Network Communications, vol. 37, pp. 24-37, 2019.

[Ve20] L. Velasco and M. Ruiz, "Supporting Time-Sensitive and Best-Effort Traffic on a Common Metro Infrastructure," in IEEE Comm. Letters, vol. 24, pp. 1664-1668, 2020.

[Za21] L. Zanzi, V. Sciancalepore, A. Garcia-Saavedra, H. D. Schotten, and X. Costa-Pérez, "LACO: A Latency-Driven Network Slicing Orchestration in Beyond-5G Networks," IEEE Trans. on Wireless Comm., vol. 20, pp. 667-682, 2021.

[Zh19] Z. Zhong, N. Hua, Zh. Yuan, Y. Li, and X. Zheng, "Routing without Routing Algorithms: An AI-Based Routing Paradigm for Multi-Domain Optical Networks," in Proc. OFC, 2019.