Intent-Based Networking and its Application to Optical Networks [Invited Tutorial]

L. VELASCO*, S. BARZEGAR, F. TABATABAEIMEHR, AND M. RUIZ

Universitat Politècnica de Catalunya, Barcelona, Spain
**Corresponding author: luis.velasco@upc.edu*

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 tutorial, we introduce the IBN paradigm and its application to optical networking, highlighting the benefits that Machine Learning (ML) algorithms can provide to IBN. Because the deployment of ML applications requires a specific orchestrator to create ML functions that are connected as ML pipelines, we show an implementation of such orchestrator. Some challenges and solutions are presented for the generation of accurate synthetic data, proactive self-configuration, and cooperative intent operation. Illustrative examples of intent-based operation and numerical results are presented and the obtained performance is discussed. © 2021 Optical Society of America

1. INTRODUCTION

Software Defined Networking (SDN) defines a centralized control plane architecture with global network vision. At the optical layer, the SDN controller can achieve optimal routing for optical connections (lightpaths) at provisioning time and during reconfiguration [1]. Besides, a distributed computing and storage infrastructure has been deployed for virtualizing network functions, which is managed by a centralized Virtual Infrastructure Orchestrator (VIO) [2]. Placed besides the SDN controller, a Monitoring and Data Analytics (MDA) controller was proposed in [3] to collect monitoring data, analyze such data, and make decisions (control loop). Such data analysis can be based on Artificial Intelligence (AI) / Machine Learning (ML) algorithms [4], which enable network automation solutions, aiming at reducing operational costs.

Among the large number of use cases for autonomous optical network operation, three major categories covering the entire lifecycle of optical connections are highlighted in [3]. The first category refers to the automation of connectivity provisioning, when the provisioning process itself requires meeting some performance, e.g., achieve resource efficiency or minimize connection blocking [5, 6]. In addition, monitoring and estimation of Quality of Transmission (QoT) is of paramount importance for both connection provisioning and reconfiguration. A second category is related to the dynamic network adaptation, which entails monitoring one or more network entities (e.g., an optical connection) and make decisions to achieve some target performance. The target is to deal with situations ranging from those that require scaling or reallocating resources to elastically adjust to demand variations in volume and direction, to those that require healing and recovery. Examples include QoT degradation and connection rerouting [7], in-operation network planning [8], dynamic capacity allocation of virtual links supported by one or more optical connections [9] or even reconfigure a virtual network topology [10, 11]. Finally, as a third category, degradation detection can be used also for failure localization [12, 13]. Here, the performance to be achieved is related, e.g., to availability metrics.

However, the drawback is the proliferation of individual control loops, which brings also complexity to network management. In addition, defining how to achieve operational goals is very complex. In this scenario, Intent-Based Networking (IBN) proposes a different approach, where intents are defined as high-level abstractions that allow network operators to define *what* are their desired outcomes, without specifying *how* they would be achieved [14]. This strategy reduces human intervention and paves the way to the application of AI/ML techniques. In an IBN environment thus, operators provide intents as inputs to guide content-based systems to implement them without human intervention. Intents allow to define the goals and outcomes and provide: *i*) data abstraction to avoid users and operators

to take care of specific device configuration; and *ii*) functional abstraction to avoid users and operators being concerned with how to achieve the goals.

Another issue is that of data availability, since AI/ML usually require a large dataset for training purposes, which is difficult to obtain. The lack of data can be compensated with the use of tools that include analytic models to explain the data plane, e.g., GNPy [15] for the optical and CURSA-SQ [16] for the packet layers. Such tools can run in sandbox domains [17] and used for training AI/ML algorithms (see [18]).

Finally, using ML algorithms for network automation entails analyzing heterogeneous data collected from monitoring points in network devices. Because network entities can be reconfigured, e.g., lightpath rerouting, it is of paramount importance to link different ML functions (i.e., performance data collection, pre-processing, analysis, storage, visualization, etc.) among them to create an *ML pipeline* and to the related network entity (e.g., a lightpath). ML pipelines are associated to network entities and need to be deployed and reconfigured using an independent orchestrator, named ML Function Orchestrator (MLFO) [19].

The rest of the paper is organized as follows. Section 2 introduces the path to network automation by reviewing previous architectures and motivating the IBN paradigm. Section 3 presents an implementation of MLFO, which includes algorithms for managing ML pipelines, including its computation and reconfiguration. An architecture and workflows are proposed, and preliminary results of an implementation are presented. Because the application of IBN is closely related to AI/ML, a short background on advanced ML techniques is provided in Section 4 to help the reader to fully understand the challenges, solutions and illustrative applications presented in Sections 5 and 6. Finally, Section 7 draws the main conclusions.

2. TOWARD NETWORK AUTOMATION

A. *Previous Architectures*

Network automation has been long time envisioned. In fact, the Telecommunications Management Network (TMN), defined by the International Telecommunication Union in [20], is a hierarchy of management layers (network element, network, service, and business management), where high-level operational goals propagate from upper to lower layers.

In the way toward autonomic adaptation to changes, while hiding intrinsic complexity to operators and users, the Internet Engineering Task Force developed the concept of Policy-Based Network Management (PBNM) [21]. PBNM separates the rules governing the behavior of a system from its functionality. In PBNM, high-level management *policies* are broken down into low-level configurations and control logic (*policy rules*) to ensure that the network provides the required services. Policies can be defined as a set of simple *control loops*; each policy rule consists of a set of events and conditions and a corresponding set of actions, where each condition defines *when* the policy rule is applicable.

The most extended PBNM architecture consists of four systems (Fig. 1a): *i*) the policy management tool allows operators to define and update policies and it translates and validates policy rules; ii) the policy repository that stores the policies; *iii*) a set of policy decision points, which interprets the policies, translates them into a device-specific representation, and triggers the execution of the related actions whenever they satisfy the specified conditions; and *iv*) the policy enforcement points running on a policy-aware node that executes the policies. The drawback of PBNM is solving conflicts that might arise within or among policies; conflict resolution requires some external system or iterations with operators and/or users.

The network management architecture has evolved with the development of the SDN concept that brings programmability to simplify configuration (it breaks down high-level service

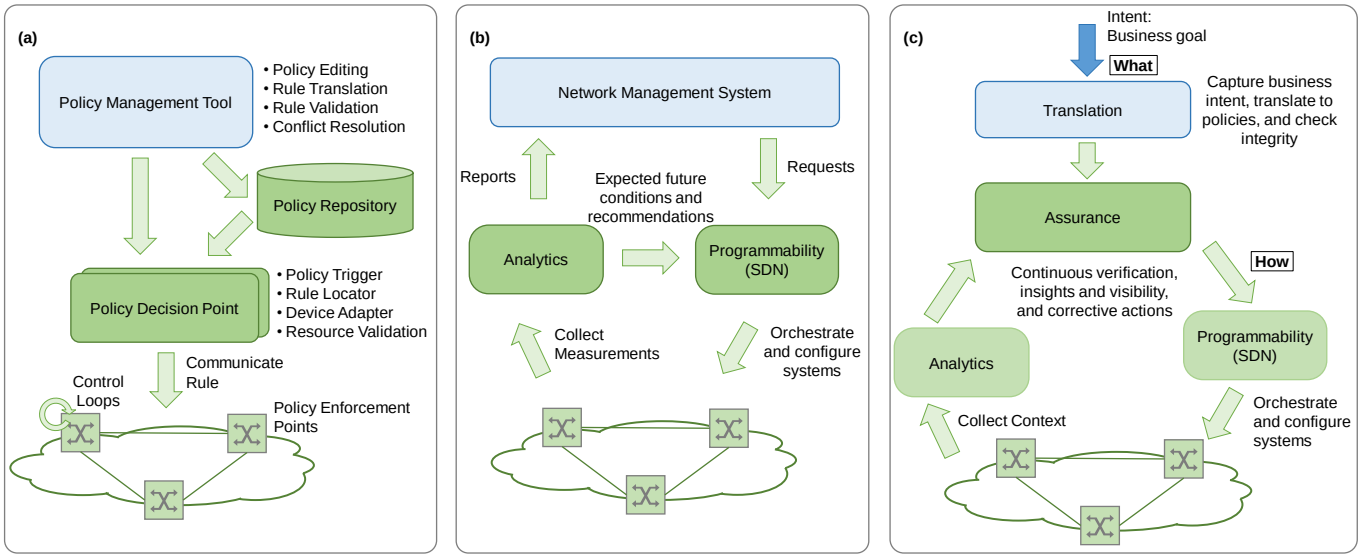

Fig. 1. Steps Toward Network Automation: Policy-Based Network Management (a), Monitoring and Data Analytics (b), and Intent-Based Networking (c).

abstraction into lower-level device abstractions), orchestrates operation, and automatically reacts to changes or events. A data analytics system [22] can complement the SDN controller (Fig. 1b), so the network becomes proactive. Being proactive is of paramount importance, as the analytics system could anticipate anomalies and degradations before they cause major problems or become failures. Upon the detection, the analytics system can issue proper recommendations to the SDN controller, which can take the most appropriate action. Additionally, such analysis can be extended to forecasting network conditions that can be used to improve resource efficiency. In this architecture, control loops can be defined at various levels, from the device [23] to the network, depending on the use case [24, 25], as monitoring is collected and can be analyzed locally and/or network-wide. The drawback of this architecture is that the analytics system needs to combine information about services and the network itself, which, in practice, requires redesigning that and other control and management systems.

B. Intents and Intent-Based Networking

IBN complements SDN control and orchestration by allowing a declarative syntax while abstracting the operational process and focusing on behavior. Service definition can be based on templates to define resources and relationships for the service and allow specifying the Intent in terms of policy rules that guide the service behavior, specifying the applications, analytics and closed control loop events needed for the elastic management of the service.

A *translation* mechanism is needed to convert the intent into a network configuration to be automatically deployed within the network infrastructure and a set of policies that the IBN needs to verify that such policies can be executed (Fig. 1c). During the service lifecycle, the service *assurance* system makes sure that the network continues to deliver on that intent based on the specified design, analytics, and policies and with the help of ML algorithms. Intent-Based ML algorithms find the right knowledge and data to identify conditions with significant semantic value (insights) from raw telemetry, without being explicitly programmed. Actionable insights and rich context together with policy-driven closed loops can take automated actions whenever the network deviates from the intent. Reporting is intended to generate descriptive outputs, e.g., statistical summaries, as well as knowledge transfer of main key performance indicators of the service. Differentiated reports can be generated, so applications can reconfigure policies to adjust to service requirements and the network management can gather knowledge transferred for different services and processed jointly to improve actions [26-28].

An ML pipeline associated to the service can be also created, as specified in [19]. The ML pipeline consists of a set of ML nodes (e.g., collectors, pre-processors, models, policies, etc.) that are combined to form an analytics function and are managed by an MLFO. The MLFO is an independent orchestrator that manages ML pipelines by placing functions in different locations in the network and manage their connectivity to create a chain. Based on ML algorithms, IBN suggests the optimal network configuration for the services and the associated ML pipeline prior to deployment. Finally, the IBN architecture can be complemented with sandbox domains, where model training will be performed with data from a data lake populated from heterogeneous data and context sources, including network, applications, and other systems, and augmented with data from simulation [29-30].

3. MACHINE LEARNING FUNCTION ORCHESTRATION

Many intent-based applications might focus on the operation of one single network entity, where the source of monitoring data and the point of actuation are closely related, e.g., a packet node or an optical transponder. However, generally speaking, IBN applications require a global view of the network resources, either to make decisions, to bring performance measurements and operation action to a global system, or both. A very natural application is for failure localization, where data needs to be collected from heterogeneous sources and analyzed together. Based on the definition in [19], in this section, we present an MLFO implementation as an independent orchestrator that manages ML pipelines by placing ML functions in different locations in the network and connecting them to create a chain [31].

A. ML Pipeline Management

Let us first focus on the ML Pipeline computation. We assume that an entity in the data plane (e.g., a lightpath) requires deploying an ML pipeline with five different ML functions: *i*) collector (*Co*) in charge of collecting monitoring data from activated monitoring points (*M*), e.g., from the end optical transponders and spectrum analyzers deployed in intermediate optical nodes [24, 25]; *ii*) aggregator (*Ag*) that collects measurements from a number of different collectors and perform some not computationally intensive task, like compute some statistics, e.g., max, min, and average; *iii*) processor (*Pr*), which performs more computational intensive task on the received data; *iv*) a time series database (*DB*); and *v*) a user interface (*UI*). The relation among those ML functions can be specified as a graph $T=(V,A)$ (see Fig. 2a).

An optimization problem can be defined to find the optimal solution, given graph *T*, the description of the set of nodes *N*, the constraints for the set of arcs *A*, as well as some other constraints, like fixed ML function placements. The output of the problem should include the placement of the ML functions to create an ML pipeline that follows template *T* and meets the constraints, while minimizing some utility function (e.g., number and capacity of containers/VMs to be deployed, connectivity cost, etc.) subject to the state of the resources in the telecom cloud infrastructure. We name such problem *ML Pipeline Deployment* (PLD) and it can formally be stated as:

Given:

- *Pipeline Template*: graph $T = (V, A)$, where V = {<allowed types={*type*}>}, A = {<maxDelay, minCapacity>}, and *type*=<container/VM descriptor, fan-out>.

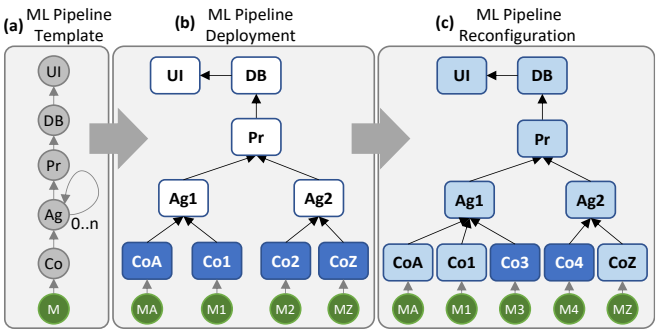

Fig. 2. Example of ML pipeline template (a), deployment (b) and reconfiguration (c).

- *Constrained nodes*: set $N = \{<n, F>\}$, where $n = type \in v$."allowed types" | $v \in V$, and F is a set of constraints, e.g., a location, and it can be empty.
- Telecom cloud infrastructure, i.e., edge/cloud computing and connectivity.

Output: The pipeline P to be deployed, i.e., $P = (N', E) \sim T$, $N' \supseteq N$ and every $e \in E$ is an instance of $a \in A$ that satisfies its constraints.

Objective: Minimize some utility function.

To solve the PLD problem, a heuristic algorithm can be developed to place the constrained ML functions and find the shortest paths connecting the already placed ML functions. During such process, more ML functions can be added to meet the given constraints until graph P following template T is obtained. An example of solution is illustrated in Fig. 2b, where the location of the collector ML functions have been specified to be in the same location as the related monitoring point. Once the PLD problem is solved, the obtained solution needs to be deployed.

Because the ML pipeline is linked to an entity in the data plane, when that entity is reconfigured, its ML pipeline might also need to be reconfigured. Therefore, we can define a different optimization problem, where we are given the template T, the current ML pipeline P and the new set of constrained nodes and the objective is to reconfigure P with the minimum cost (e.g., number of changes in P' with respect to P and the total cost, etc.). We name such problem *ML Pipeline Reconfiguration* (PLR) and it can be stated as:

Given:

- The template T, the deployed pipeline P and the new set of constrained nodes N.
- Telecom cloud infrastructure, i.e., edge/cloud computing and connectivity.

Output: The new pipeline P' to be deployed.

Objective: Minimize some utility function.

A heuristic similar as the one for the PLD problem can be devised, where the deployed ML functions that are not in the new constrained set of nodes are first removed, and the nodes that need to be migrated are disconnected and moved. An example of reconfiguration is presented in Fig. 2c, where ML function Co2 is not needed for the new ML pipeline, and Co3 and Co4 have been added.

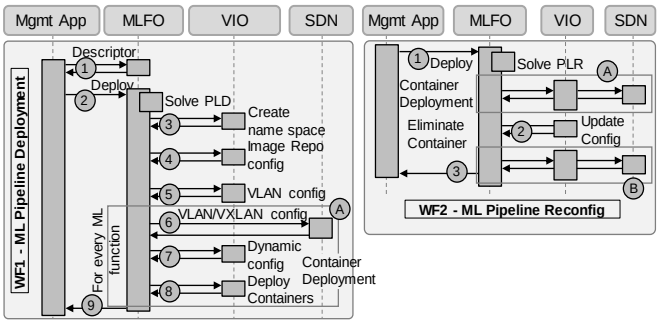

Fig. 3. ML pipeline deployment (WF1) and reconfiguration (WF2).

B. *Proposed Architecture and Workflows*

In our architecture, an external management application system triggers ML pipeline deployment. Therefore, the MLFO needs to expose a Northbound Interface (NBI) to receive the description of the ML pipeline, including the template, constrained nodes, etc. (collectively named ML pipeline *descriptor*). The MLFO runs the PLD and PLR optimization problems to compute the ML pipeline to be deployed (*deployment plan*), and coordinates with the VIO and the packet layer SDN controller. A specific VLAN is created for each ML pipeline for intra-DC communications, whereas we assume that connectivity between two DCs is based on pre-established connections (e.g., VXLAN tunnels).

Fig. 3 presents the workflows for the initial ML pipeline deployment and for any subsequent externally-triggered reconfiguration. Let us start with the deployment workflow (WF1). The management application initiates WF1 by sending the deployment plan (WF1 message 1 in Fig. 3). The descriptor contains a template for the ML functions and for the connectivity. Next, the deployment is triggered (2) and the MLFO starts a series of steps. First, the MLFO solves the PDL using the constraints, resulting in a mapping between the ML functions and the datacenters, and the connectivity and the deployment plan is computed. A list of iterations is generated that includes the communication of the MLFO with the VIO (e.g., Kubernetes) for the deployment of the ML functions (e.g., encapsulated into containers), and with the SDN controller for managing the connectivity among the ML functions. The list iterations include: *i*) the namespace creation (3); *ii*) the configuration of an image repository storing the different computing images that are retrieved when a new ML function instance is deployed (4); *iii*) the configuration of the ML pipeline network that entails creating the VLAN (5) and pairing it to the VXLAN tunnels (6); *iv*) the creation of a volume for the dynamic configuration of the computing instances (7); and *v*) the deployment of the containers (8). Steps 6-8 are followed for every ML function to be deployed (block A). A reply is eventually sent (9).

WF2 is triggered when the ML pipeline needs to be reconfigured. The management application initiates WF2 by sending a new set of constraints to the MLFO (WF2 message 1 in Fig. 3). The PLR problem is then solved considering the received configuration. Then, the MLFO finds the changes to

be performed and prepares a plan with the creation of new ML functions (block A) and removal of existing ones (block B). Besides, the dynamic configs are updated to reflect the changes in the system (e.g., changes on the IPs) (2). When all the steps are executed, the result is sent back to the management application (3).

C. Implementation and results

We implemented the MLFO in Python 3.8 that exposes a REST API NBI. Kubernetes was used as VIO and docker as the container technology [32]. The MLFO uses the Kubernetes API through a python client library. A private image repository was hosted in Docker Hub. Multus and Open vSwitch Container Network Interface plugins were used for the VLAN configuration thorough Kubernetes. Kubernetes ConfigMap was used for the dynamic configuration files mounted into the containers as read-only files. The ingress-nginx controller was used for the reverse proxy; the configuration is done through the Kubernetes' ingress resource, which exposes a service through a load balancer. As for the SDN controller, we used OpenDayLight, which controls intra-DC switches through the OpenFlow protocol. VXLAN was used as tunneling technology for inter-DC tunnels. The MLFO pairs VLANs with VXLANs on each of the intra-DC switches that are involved on a connection between two containers.

To summarize the results, the time to run WF1 to deploy the ML pipeline in Fig. 2b was about 1 min, whereas that of WF2 to reconfigure the ML pipeline as in Fig. 2c was 8.5 sec.

4. BACKGROUND ON ADVANCED ML TECHNIQUES

Many intent-based solutions need from ML techniques as a way to implement proactive approaches. In this section, we give some background on advanced ML techniques that are used in the applications that are presented in the next sections. Note that simpler (but not necessarily less effective) ML techniques for network automation can be found in [4].

A. Regression for Time Series

Time series forecasting covers those methodologies that predict future events as a function of previous observations, as well as some additional features that may or not depend on time. Traditionally, Autoregressive Integrated Moving Average (ARIMA) models [33] have been proposed for time series forecasting, due to several key characteristics, such as easiness of interpretability and the ability to provide probability distributions of the predicted events. They assume linearity between features and need some data pre-processing to remove important components out of the model (such as trend or heteroscedasticity), which reduces their applicability for more complex time series events.

Deep learning techniques can be applied to predict complex future events without considering strongly limiting assumptions. In particular, the use of feed-forward neural networks (FFNN) [34] 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.

In general, FFNNs work better with pre-processed features that summarize the input information to be considered for prediction, e.g., some statistics and trend of the last observed events. This can be a limiting factor if features are not well designed. Another approach is to use raw data, e.g., all data observed in a large past window. In this regard, convolutional neural networks (CNN) have the inherent ability to learn and automatically extract features from raw input data [35]. By means of hidden convolutional layers, automatic identification and extraction of relevant features is produced in an unsupervised manner.

Although both FFNN and CNN can be designed and trained to predict time series events, they were devised for applications that do not depend on time. On the contrary, Recurrent Neural Networks (RNN) [36] 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. To improve RNNs, Long Short-Term Memory (LSTM) networks [36] 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 [37].

B. Reinforcement Learning

Reinforcement Learning (RL) considers the paradigm of an intelligent *agent* that takes actions in an *environment* (as in Fig. 5c). At every discrete time step t, with a given state s, the agent selects action a with respect to a policy, and it receives from the environment a reward r and the new state s'. The objective is to find the optimal policy that maximizes a cumulative reward function. RL fits perfectly as part of intent agents, as the related problems can be usually stated in the form of a Markov decision process and they can be solved RL using dynamic programming techniques. In addition, in contrast to supervised learning, RL does not need labeled datasets and it can correct sub-optimal actions through exploration.

The simplest RL is Q-learning [38], which is a model-free discrete RL method that uses a Q-table to represent the learned policy, where every pair $<s, a>$ contains a q value. Being at state s, the action a to be taken is the one with the highest q value (or it is chosen randomly). Once the action is implemented and the new state s' and the gained reward r are received from the environment, the agent updates the corresponding q value in the Q-table. Q-learning works efficiently for problems where both states and actions are discrete and finite. However, it usually introduces overestimation, which leads to suboptimal policies, and the Q-table grows with the number of states.

Deep Q-learning (DQN) substitutes the Q-table by a FFNN that receives a continuous representation of the state and returns the expected *q* value for each discrete action [38]. However, the FFNN tends to make learning unstable, so a *replay buffer* can be used to retrain the FFNN. *Double DQN* [39] uses two different FFNNs (*learning* and *target*) to avoid overestimation, which happens when a non-optimal action is quickly biased (due to noise or exploration) with a high *q* value that makes it preferably selected. The learning model is updated using the *q* values retrieved from the target model, which is just a simple copy of the learning model and it is periodically updated. Finally, *dueling double DQN* (D3QN) [40] uses two different estimators to compute the *q* value of a pair <*s*, *a*>: *i*) the *value* estimator, an average *q* value of any action taken at state *s*; and *ii*) the *advantage* estimator, which is the specific state-action dependent component. The sum of both components returns expected *q* values.

DQN-based methods assume a finite discrete action space. Nonetheless, other approaches, such as *Actor-Critic* methods [41], use continuous state and action spaces. *Actor-Critic* methods train two different types of models separately: *i*) *actors*, which compute actions based on states, and *ii) critics* that evaluate the actions taken by actors, i.e., compute *q* values. Both actor and critic models can be implemented by means of FFNNs. Aiming at reducing overestimation, the Twin Delayed Deep Deterministic Policy Gradient (TD3) method [41] considers one single actor and two different critic models, where the minimum value from the two critics is used for learning the optimal policy.

5. SOME CHALLENGES AND POSSIBLE SOLUTIONS

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, QoT measurements depend on the actual routing and spectrum allocation of an optical connection; in consequence, real measurements can only be available after such optical 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.

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 (see Section 2). 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.

The next subsections present possible solutions for these challenges.

A. Generation of reliable and accurate synthetic data

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.

To illustrate accurate data generation, Fig. 4 presents two examples of digital twins for the packet (Fig. 4a-b) and the optical layer (Fig. 4c-d). For the packet layer, Fig. 4a presents an example of a network with four nodes interconnecting four data centers (DC), 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. 4b based on the CURSA-SQ methodology [16]. CURSA-SQ's 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. In the case of the optical layer, Fig. 4c reproduces an example of optical connection established between two locations A and Z; optical transponders in the remote locations, cross-connects and intermediate amplifiers are represented. As for the optical layer, a digital twin is represented in Fig. 4d, based on the GNPy tool [15] to estimate the expected Signal to Noise Ratio (SNR) of the optical connections. In this case, the dynamic configuration module finds the most likely value of modeling parameters based on the monitoring data received from the network (see [12]).

B. Proactive Self-configuration

Autonomous network operation can be *reactive* (i.e.,

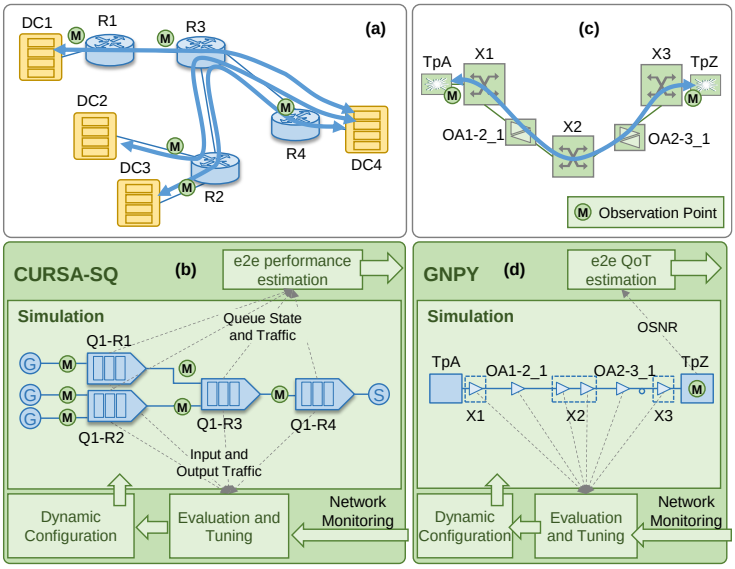

Fig. 4. Examples of digital twins for the packet (a-b) and the optical (c-d) layers.

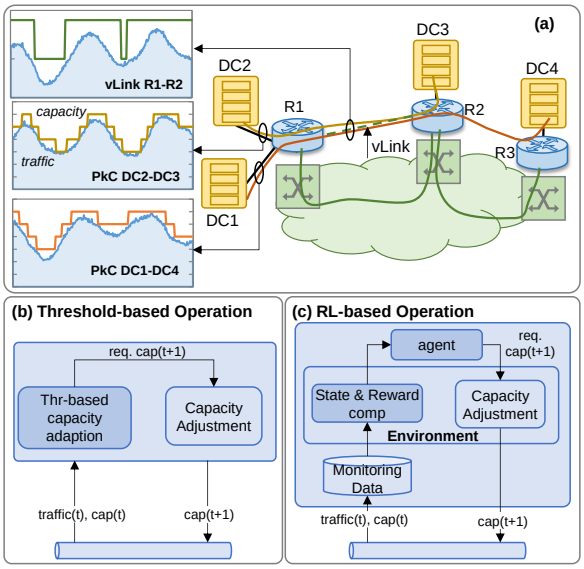

Fig. 5. Capacity operation of PkCs and vLinks.

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.

When PkCs are routed on top of virtual networks, where virtual links (*vLink*) are supported by the optical layer, capacity might not be instantly allocated. Let us illustrate this problem with an example. Fig. 5a 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 or activating and deactivating subcarriers in Digital Subcarriers Multiplexing (DSCM) systems [9].

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. 5a 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. 5b-c present two alternative approaches to operate the capacity of the PkCs, based on a simple threshold rule or based on an intelligent ML-based algorithm, in this case, RL. Every connection (PkC or vLink) 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.

C. *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. 5a. 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.

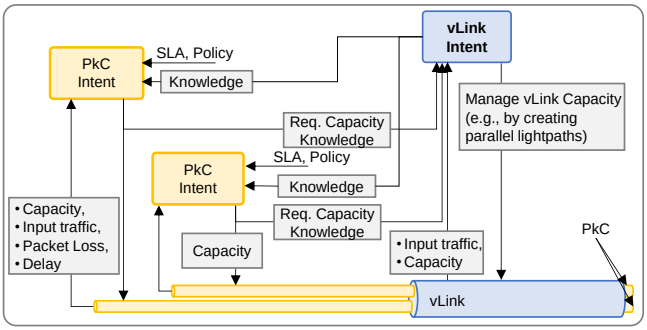

Fig. 6. Intent cooperation and transfer knowledge.

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. 5a. 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. 5a 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. 6 supports a hierarchy of intents, where each intent agent is in line with that in Fig. 5b-c. 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.

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. Finally, knowledge transfer would benefit directly from the MLFO introduced in Section 3, as it would facilitate the dynamic association between PkC and vLink intents.

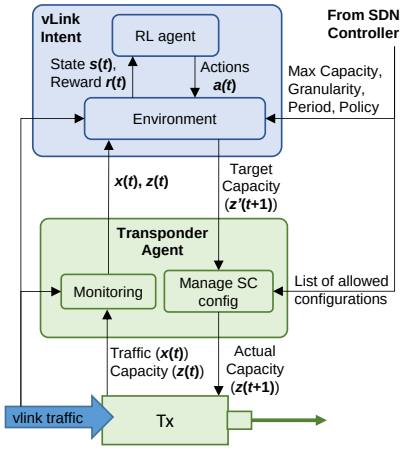

Fig. 7. RL-based Autonomous vLink Operation Architecture.

6. ILLUSTRATIVE INTENT-BASED APPLICATIONS

In this section, we present two illustrative examples of intent applications based on ML. In the first example, the proactive self-configuration solution presented in Section 5.B is extended for managing the capacity of the vLink. In this approach, the vLink intent takes actions based on the traffic in the vLink. We go a step beyond in the second application, where intent cooperation is showcased developing the ideas introduced in Section 5.C. 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. Autonomic vLink Capacity Adaptation

Let us now illustrate the application of RL techniques for the autonomous operation of the capacity of a vLink. To this aim, we extend the RL-based intent agent in Fig. 5c with the architecture in Fig. 7; a RL-based vLink intent analyzes monitoring data, 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. With such capacity, an agent running at the optical transponder (Tx side) decides the *actual capacity* $z(t+1)$ to be allocated, which will depend on the characteristics of the optical layer. The operation can be based on policies and other parameters received from the SDN controller.

Without loss of generality, let us consider that the optical connection supporting the vLink is based on DSCM. The key-aspect of DSCM is the use of multiple subcarriers (SC), where each SC can be activated/deactivated and configured independently of the others in terms of modulation format and symbol rate. In addition, dynamic capacity adaptation can be carried out at the data plane without control plane intervention; the Tx side decides the configuration and activates a SC, and the Rx automatically detects the new SC and determines its configuration [9].

Autonomic RL-based operation can be configured to achieve a desired operational goal maximizing the long-term reward by taking proper actions to the environment. In this illustrative application, the goal is to adjust the vLink capacity to guarantee that the vLink load $l(t)$ (defined as $x(t) / z(t)$) does not exceed but is close to a given maximum l_{max}; this will minimize over-provisioning (defined as $z(t)$-$x(t)$) while limiting the average maximum delay for the traffic. In this scenario, the learning process can be focus on the traffic variation and its evolution with time, which is key for a tight load adjustment and for avoiding high delay and traffic loss due to insufficient capacity allocation.

The environment in the intent agent is in charge of computing the state $s(t)$ based on the approach in [9], whereas $s(t)$ is obtained as a function of both $l(t)$ and l_{max}. Given $s(t)$, the learned policy performs action $a(t)$, which consists in a capacity volume Δz to be added to or subtracted from the current vLink capacity. The reward function $r(t)$ is a linear function with three penalty components (ordered by importance): *i*) traffic loss ($x(t) > z(t)$), *ii*) l_{max} violation ($l(t) > l_{max}$), and *iii*) over-provisioning ($z(t)$ - $x(t)$). Thus, the maximum reward is achieved when neither loss nor l_{max} violation is observed, and over-provisioning is minimized.

A Python-based simulator reproducing the architecture in Fig. 7 has been implemented for evaluation purposes. Realistic vLink input traffic was generated using the flow simulator and parameters described in [16], which resulted into a daily traffic pattern varying between 20 and 240 Gb/s. We assume that the DSCM system consists of 8 SCs, where each SC can be configured with either 8QAM or 16QAM at 11 Gbaud to serve the target capacity determined by the vLink intent agent. The target load l_{max} was set to 80%.

Q-learning, D3QN, and TD3 RL methods were implemented, adapting state and action spaces to either discrete or continuous space depending on the method (see Section 5.B). The FFNNs for D3QN and TD3 methods were configured with 2 layers each with 100 neurons implementing ReLU activation function [34]. For the sake of fairness, Δz was setup to 10 Gb/s in all the methods. In addition, a threshold-based approach was implemented for benchmarking purposes, which reactively adds or releases capacity to keep $l(t)$ in the range [0.7-0.8].

Fig. 8 shows the performance of the threshold-based approach and RL-based methods (note that latter ones need a sufficiently large number of episodes to guarantee robust and efficient operation). The measured input traffic $x(t)$, target capacity $z'(t)$ in the case of RL-based methods, and allocated capacity $z(t)$ are plotted for a typical day. We observe that the threshold-based method with an *a priori* good configuration produces poor performance (traffic loss and high delay) as a result of its reactive nature. In contrast, all RL-based methods learned policies to avoid losses and they adapt better to the traffic characteristics. Specifically, D3QN achieves low maximum delay (at least, half of the other RL methods). However, the main conclusions to be extracted from delay analysis is that although the reward function explicitly controls the load, which is closely related to delay performance, a finer delay control (e.g., keeping delay below a target maximum) requires *ad-hoc* delay analysis components to be considered in the reward function. Table 1 summarizes the results. In general, RL methods require higher overprovisioning than the threshold-based approach, but such overprovisioning is necessary to achieve the target performance. Therefore, these results validate the usefulness of RL for the proposed vLink operation use case.

Table 1. vLink Capacity Adaptation Summary

Method	Loss (Gb/s)	Over-Provisioning (Tb/day)	Num SC Changes	Delay (μs)		
				min	avg	max
Thr-based	4.05	19.2	**18**	45	129	1629
Q-Learning	0	21.6	54	23	95	603
D3QN	0	22.6	34	**19**	**82**	**267**
TD3	0	20.8	70	21	96	512

The drawbacks rely on the need for a larger number of SC changes (activations and deactivations) compared to the threshold-based approach. E.g., among the RL-based methods, the lowest overprovisioning is achieved by TD3 at the expense of doubling the number of SC changes with respect to D3QN, which requires double number of SC changes than the threshold-based approach. Hence, the selection of the RL method is not trivial and it might depend on limitations of the hardware, e.g., the SC activation time.

Finally, it is worth noting that RL-based operation at the vLink level cannot provide differentiated delay performance for the different PkCs supported by the vLink. On the contrary, implementing the RL-based operation at the PkC would provide specific performance to the individual PkC but would require from specific cooperation between PkC and vLink intent agents.

B. Cooperative Intent Operation

In this section, we extend the previous stand-alone intent-based vLink capacity adaptation and focus on evaluating the potential benefits of the hierarchical cooperation introduced in Section 5.C. Fig. 9a shows the architecture, where PkC intent agents implement a RL-based method similarly to the previously used for the vLink. For PkCs specifically, let us consider different operational goals than those used for the vLink; every PkC has a different requirement in terms of maximum delay budget d_{max} that needs to be guaranteed. This entails changes in the reward function $r(t)$, where a new component adds a large penalty if the measured delay in the packet connection exceeds the required d_{max}. Moreover, no penalty for maximum load violation is considered.

Fig. 9a shows the hierarchical cooperation between PkC and vLink intents. 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 [42], a LSTM-based traffic model can predict the evolution of the traffic and used to define the state

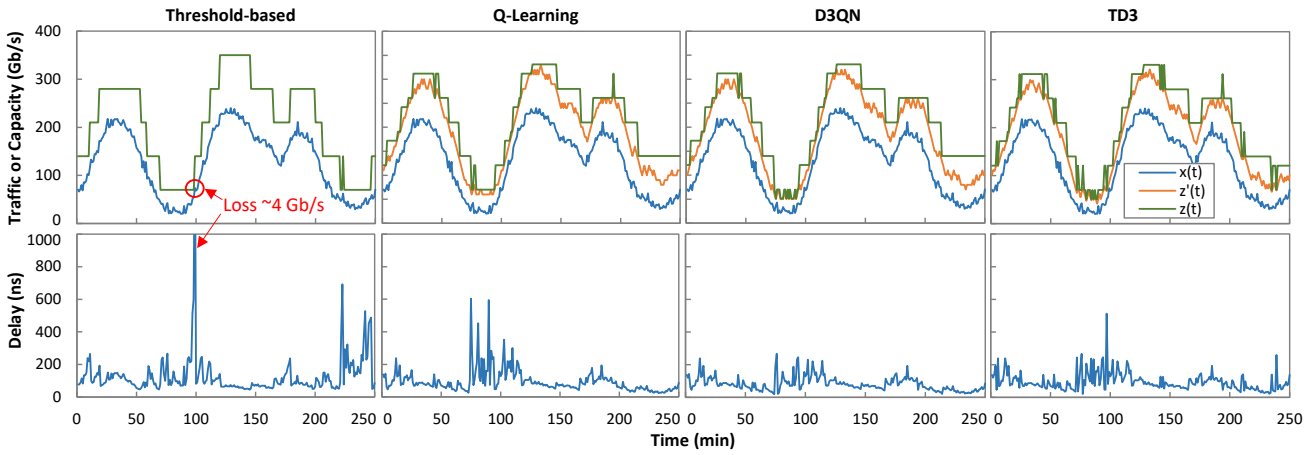

Fig. 8. Autonomic vLink Capacity Adaptation (a) and obtained delay (b).

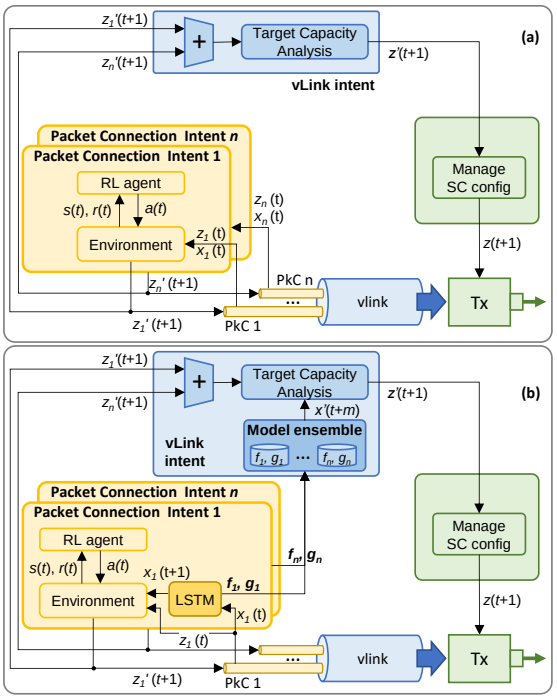

Fig. 9. Extended architecture with hierarchical intent cooperation (a) and knowledge transfer (b).

for the RL agent. 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. 9b). 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. 10a-b 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. 10a); 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. 10b). 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.

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 $\max(x'(t+m), z'(t+1))$.

The numerical evaluation scenario detailed in Section 6.A is adapted to fit this example. Three PkCs A, B, C with maximum traffic 120, 60, and 60 Gb/s and different delay budgets have been considered to generate the same aggregated vLink traffic as in Fig. 8. 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. We focus on the TD3 RL method and similarly as for vLink intent operation, we run it until achieving a robust and accurate operation.

The accuracy of the proposed transfer knowledge scheme to predict vLink traffic is illustrated in Fig. 10c. 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. 10d shows the delay at the source node for all PkCs, assuming that PkCs leave through 100/200

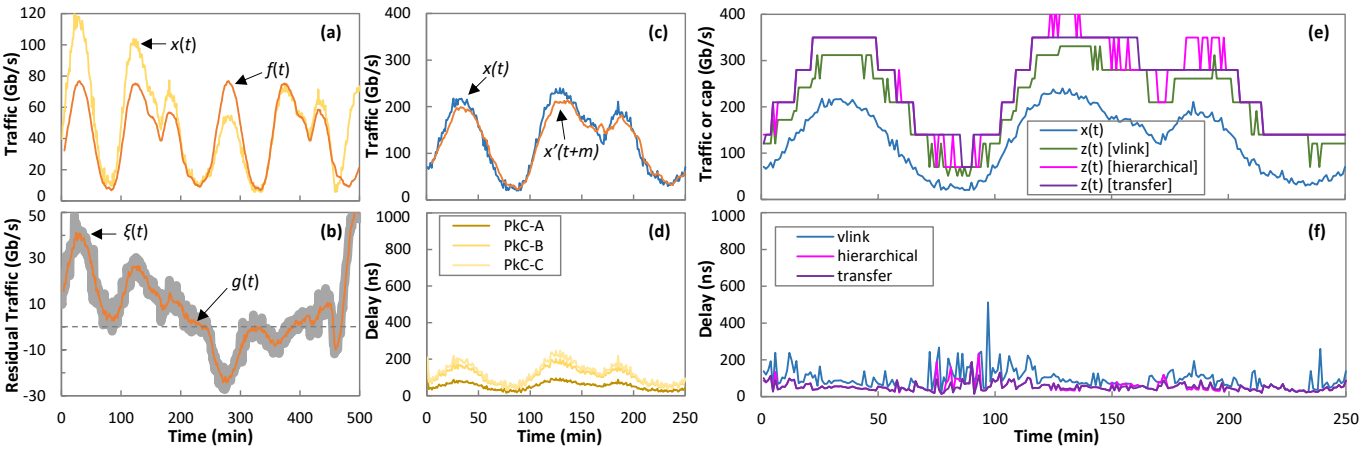

Fig. 10. Compound traffic model for PkCs (a-b), PKC-vLink intent cooperation performance (c-d), and comparative results (e-f).

Table 2. 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**

Gb/s interfaces. We observe that RL-based operation at the PkC level allows achieving the targeted differentiated delay performance.

From Fig. 10c-d, 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. 10e-f and Table 2 compare hierarchical cooperation without and with knowledge transfer, named *hierarchical* (Fig. 9a) and *transfer* (Fig. 9b), respectively. For comparison purposes, RL-based vLink intent (Fig. 7) performance is considered. Results of the vLink allocated capacity and introduced delay are presented in Fig. 10e and Fig. 10f, 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. CONCLUDING REMARKS

In this tutorial, the IBN paradigm was introduced focused on its application to optical networking. 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. First the deployment of ML applications requires from a specific orchestrator, named MLFO, to create ML functions that are connected as ML pipelines. An implementation of MLFO for the deployment and reconfiguration of an ML pipeline related to a lightpath was shown. Some challenges and solutions were afterwards presented for the generation of reliable and accurate synthetic data by using digital twins running in sandbox domains, proactive self-configuration, and cooperative intent operation and transfer knowledge. Illustrative examples of intent-based operation that use ML techniques were introduced. First, intent agents based on RL were proposed to adjust the capacity of a vLink as a function of the input traffic. Second, a combined LSTM and RL approach was proposed 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.

Funding. The research leading to these results has received funding from the MICINN IBON (PID2020-114135RB-I00) project and from the ICREA Institution.

REFERENCES

1. L. Velasco, D. King, O. Gerstel, R. Casellas, A. Castro, V. López, "In-Operation Network Planning," IEEE Communications Magazine, vol. 52, pp. 52-60, 2014.
2. M. Bonfim, K. Dias, S. Fernandes, "Integrated NFV/SDN Architectures: A Systematic Literature Review," ACM Computing Surveys, vol. 51, pp. 1-39, 2019.
3. L. Velasco, A. Chiadò Piat, O. González, A. Lord, A. Napoli, P. Layec, D. Rafique, A. D'Errico, D. King, M. Ruiz, F. Cugini, R. Casellas, "Monitoring and data analytics for optical networking: benefits, architectures, and use cases," IEEE Network Magazine, vol. 33, pp. 100-108, 2019.

4. 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.
5. M. Dallaglio, A. Giorgetti, N. Sambo, L. Velasco, P. Castoldi, "Routing, Spectrum, and Transponder Assignment (RSTA) in Elastic Optical Networks," IEEE/OSA J. of Lightwave Technology, vol. 33, pp. 4648-4658, 2015.
6. X. Chen, B. Li, R. Proietti, H. Lu, Z. Zhu, S. J. Ben Yoo, "DeepRMSA: A deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks," IEEE/OSA J. Lightwave Technology, vol. 37, pp. 4155-4163, 2019.
7. L. Velasco, A. Sgambelluri, R. Casellas, Ll. Gifre, J.-L. Izquierdo-Zaragoza, F. Fresi, F. Paolucci, R. Martínez, E. Riccardi, "Building Autonomic Optical Whitebox-based Networks," IEEE/OSA J. of Lightwave Technology, vol. 36, pp. 3097-3104, 2018.
8. L. Velasco, A. P. Vela, F. Morales, M. Ruiz, "Designing, Operating and Re-Optimizing Elastic Optical Networks," (Invited Tutorial) IEEE/OSA J. of Lightwave Technology, vol. 35, pp. 513-526, 2017.
9. L. Velasco, S. Barzegar, D. Sequeira, A. Ferrari, N. Costa, V. Curri, J. Pedro, A. Napoli, M. Ruiz, "Autonomous and Energy Efficient Lightpath Operation based on Digital Subcarrier Multiplexing," IEEE J. on Selected Areas in Communications, vol. 39, pp. 2864-2877, 2021.
10. F. Morales, Ll. Gifre, F. Paolucci, M. Ruiz, F. Cugini, P. Castoldi, L. Velasco, "Dynamic core VNT adaptability based on predictive metro-flow traffic models," IEEE/OSA J. Optical Comm and Networking, vol. 9, pp. 1202–1211, 2017.
11. L. Velasco, Ll. Gifre, J.-L. Izquierdo-Zaragoza, F. Paolucci, A. P. Vela, A. Sgambelluri, M. Ruiz, F. Cugini, "An Architecture to Support Autonomic Slice Networking [Invited]," IEEE/OSA J. of Lightwave Technology, vol. 36, pp. 135-141, 2018.
12. S. Barzegar, M. Ruiz, A. Sgambelluri, F. Cugini, A. Napoli, 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.
13. B. Shariati, M. Ruiz, J. Comellas, L. Velasco, "Learning from the Optical Spectrum: Failure Detection and Identification [Invited]," IEEE/OSA J. of Lightwave Technology, vol. 37, pp. 433-440, 2019.
14. A. Clemm, L. Ciavaglia, L. Granville, J. Tantsura, "Intent-Based Networking - Concepts and Definitions," IRTF draft work-in-progress, 2021.
15. M. Filer, M. Cantono, A. Ferrari, G. Grammel, G. Galimberti, V. Curri, "Multi-Vendor Experimental Validation of an Open Source QoT Estimator for Optical Networks," IEEE/OSA J. of Lightwave Technology, vol. 36, pp. 3073-3082, 2018.
16. M. Ruiz, F. Coltraro, 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.
17. 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 J. of Lightwave Technology, vol. 38, pp. 3180-3189, 2020.
18. "Framework for data handling to enable ML in future networks including IMT-2020," ITU-T Y.3174, 2020.
19. "Unified architecture for machine learning in 5G and future networks," Focus group on Machine Learning for Future Networks including 5G, ITU-T, 2019.
20. "Principles for a Telecommunications Management Network," ITU-T, Rec. M.3010, 2000.
21. J. Strassner, *Policy-Based Network Management: Solutions for the Next Generation*, Morgan Kaufmann, 2004.
22. Ll. Gifre, J.-L. Izquierdo-Zaragoza, M. Ruiz, L. Velasco, "Autonomic Disaggregated Multilayer Networking," IEEE/OSA J. of Optical Communications and Networking, vol. 10, pp. 482-492, 2018.
23. A. P. Vela, M. Ruiz, L. Velasco, "Distributing Data Analytics for Efficient Multiple Traffic Anomalies Detection," Elsevier Computer Communications, vol. 107, pp. 1-12, 2017.
24. A. P. Vela, B. Shariati, M. Ruiz, F. Cugini, A. Castro, H. Lu, R. Proietti, J. Comellas, P. Castoldi, S. J. Ben Yoo, L. Velasco, "Soft failure localization during commissioning testing and lightpath operation," IEEE/OSA J. of Optical Comm and Networking, vol. 10, pp. A27-A36, 2018.
25. A. P. Vela, M. Ruiz, F. Fresi, N. Sambo, F. Cugini, G. Meloni, L. Potí, L. Velasco, P. Castoldi, "BER Degradation Detection and Failure Identification in Elastic Optical Networks," IEEE/OSA J. of Lightwave Technology, vol. 35, pp. 4595-4604, 2017.
26. X. Chen, R. Proietti, C-Y Liu, S. J. Ben Yoo, "A Multi-Task-Learning-based Transfer Deep Reinforcement Learning Design for Autonomic Optical Networks," IEEE J. on Sel. Areas in Communications, vol. 39, pp. 2878-2889, 2021.
27. M. Ruiz, F. Tabatabaeimehr, L. Velasco, "Knowledge Management in Optical Networks: Architecture, Methods and Use Cases [Invited]," IEEE/OSA J. of Optical Comm and Networking, vol. 12, pp. A70-A81, 2020.
28. F. Tabatabaeimehr, M. Ruiz, C.-Y. Liu, X. Chen, R. Proietti, S. J. Ben Yoo, L. Velasco, "Cooperative Learning for Disaggregated Delay Modeling in MultiDomain Networks," IEEE Transactions on Network and Service Management, vol. 18, pp. 3633-3646, 2021.
29. L. Velasco, B. Shariati, F. Boitier, P. Layec, M. Ruiz, "A Learning Life-Cycle to Speed-up Autonomic Optical Transmission and Networking Adoption," in IEEE/OSA J. of Optical Comm and Networking, vol. 11, pp. 226-237, 2019.
30. A. Bernal, M. Richart, M. Ruiz, A. Castro, L. Velasco, "Near Real-Time Estimation of End-to-End Performance in Converged Fixed-Mobile Networks," Comp. Comm., vol. 150, pp. 393-404, 2020.
31. A. Wassington, L. Velasco, Ll. Gifre, M. Ruiz, "Implementing a Machine Learning Function Orchestration," accepted in ECOC, 2021.
32. D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," IEEE Cloud Computing, vol. 1, pp. 81-84, 2014.
33. R. Shumway and D. Stoffer, *Time Series Analysis and Its Applications*, Springer International Publishing, 2017.
34. P. Zhang and M. Qi, "Neural network forecasting for seasonal and trend time series," European J. of Operational Research, vol. 160, pp. 501-514, 2005.
35. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*, Springer, 2018.
36. D. Mandic and J. Chambers, *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*, Wiley, 2001.
37. V. Le Guen and N. Thome, "Shape and Time Distortion Loss for Training Deep Time Series Forecasting Models," In Proc. NeurIPS, 2019.
38. R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd Ed., MIT Press, 2018.
39. H. Hasselt, A.Guez, D. Silver, "Deep Reinforcement Learning with Double Q-learning," in Proc. AAAI Conference on AI, 2016.
40. Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," in Proc. ICML, 2016.
41. S. Fujimoto, H. van Hoof, D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in Proc. ICML, 2018.
42. F. Tabatabaeimehr, S. Barzegar, M. Ruiz, L. Velasco, "Combining Long-Short Term Memory and Reinforcement Learning for Improved Autonomous Network Operation," in Proc. OFC, 2021.