


# Scalability analysis of machine learning QoT estimators for a cloud-native SDN controller on a WDM over SDM network

C. MANSO,<sup>1,\*</sup>  R. VILALTA,<sup>1</sup>  R. MUÑOZ,<sup>1</sup>  N. YOSHIKANE,<sup>2</sup> R. CASELLAS,<sup>1</sup>   
R. MARTÍNEZ,<sup>1</sup>  C. WANG,<sup>2</sup> F. BALASIS,<sup>2</sup> T. TSURITANI,<sup>2</sup>  AND I. MORITA<sup>2</sup>

<sup>1</sup>Optical Networks and Systems Department, Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), 08860 Castelldefels, Spain

<sup>2</sup>KDDI Research, Inc., 2-1-15 Ohara, Fujimino-shi, Saitama, 356-8502, Japan

\*Corresponding author: carlos.manso@cttc.es

Received 18 November 2021; revised 3 February 2022; accepted 3 February 2022; published 25 February 2022 (Doc. ID 449009)

Maintaining a good quality of transmission (QoT) in optical transport networks is key to maintaining the service level agreement between the user and the service provider. QoT prediction techniques have been used to assure the quality of new lightpaths as well as that of the previously provisioned ones. Traditionally, two different approaches have been used: analytical methods, which take into account most physical impairments that are accurate but complex, and high margin formulas, which require much less computational resources at the cost of high margins. With the recent progress of machine learning (ML) together with software defined networking (SDN), ML has been considered as another option that could be both accurate and that does not consume as many resources as analytical methods. SDN architectures are difficult to scale because they are usually centralized; this is even worse with QoT predictors using ML. In this paper, a solution to this issue is presented using a cloud-native architecture, and its scalability is evaluated using three different ML QoT predictors and experimentally validated in a real wavelength-division multiplexing (WDM) over spatial-division multiplexing (SDM) testbed. © 2022 Optica Publishing Group under the terms of the [Optica Open Access Publishing Agreement](https://doi.org/10.1364/JOCN.449009)

<https://doi.org/10.1364/JOCN.449009>

## 1. INTRODUCTION

Traditional transport networks have been built upon dedicated devices that need to be manually configured when a new configuration or a new device is introduced into the network. 5G networks are a reality nowadays, and together with Internet of Things (IoT) services, they force the networks to adapt faster and to be more resilient to errors. Software defined networking (SDN) has been consolidated in recent years as a technology that addresses those challenges, based on the decoupling of the data plane that is responsible for the forwarding of the network packets, and the control plane, which makes the decision of where to forward the packets. Since the control plane and the data plane are decoupled, new application programming interfaces (APIs) are needed to adapt to this new strategy. The northbound interface (NBI) is used by applications to communicate to the SDN controller, while the southbound interface (SBI) is used by the SDN controller to communicate with the data plane. SDN architectures are usually based on open standards and APIs instead of using proprietary interfaces for the control plane, such as OpenFlow [1] for the SBI or Transport API [2] (TAPI) for the NBI. This standardization has enabled the centralization of the control plane, aggregating multiple

domains, even multiple layers, over the same SDN controller. This makes the establishment of end-to-end connections much easier, faster, and more flexible, due to the automation that SDN brings, allowing for less expenditure on human resources and thus lower operating expenditures. While the centralization has brought many advantages, the emergence of novel optical technologies such as the elastic optical network (EON) or spatial-division multiplexing (SDM) has increased the complexity of the network that needs to be managed. The degrees of freedom that these technologies allow require very high computational resources to calculate the optimum parameters, such as modulation format, channel spacing, or symbol rates.

Predicting the QoT of unestablished optical signals is key to guaranteeing the service level agreement (SLA) of the optical path. Physical layer impairments have to be taken into account such as polarization effects, dispersion, or phase shifts. Nonlinear impairments are especially difficult to estimate, and analytical models like the split-step Fourier method [3] introduce high computational costs that are hard to manage; thus, it is unfeasible in a large-scale deployment. On the other hand, high margin models {based, for example, on the Gaussian noise (GN) model [4]}, are based on power budget estimations that

usually assume the worst-case scenario on different nonlinear effects, reducing complexity at the cost of accuracy.

Machine learning (ML) offers a new way to predict the QoT of optical paths. After the ML model has been trained, it is able to predict an optical performance metric, such as the bit error ratio (BER) or Q factor, from the description of the requested connection (modulation format, wavelength, spatial path, etc.). Although QoT ML techniques do not consume as many resources as analytical methods, it is still unfeasible to rely on them for large-scale deployments because of the centralization of the control plane on SDN, where the SDN controllers have to manage a large number of network elements at the same time and hence lots of connections coming from those domains. As a result, a lot of stress is put on the controllers that are usually developed as monolithic applications [5].

Cloud-native applications rely on an architecture design with cloud computing in mind. The cloud computing paradigm offers the pooling of shared computing resources that can serve multiple tenants dynamically [6]. The applications are based on microservices, which are a set of loosely coupled isolated services that interact with each other by means of lightweight protocols, such as Google Remote Procedure Calls (gRPC) [7]. These microservices are deployed within containers, a type of virtualization mechanism that serves an isolated environment from the rest of the operating system [8]. They come with little overhead when compared to virtual machines that make use of a hypervisor, which shares the resources between the full operating system they contain and the hardware. On the other hand, containers, like Docker [9] or Linux Containers (LXC) [10], do not need to contain the full operating system; instead, they make use of the guest operating system containing only the necessary libraries for the application. This approach allows easier development, higher resilience against infrastructure problems, the ability to swap or update components on the fly, and higher scalability. Providing automatic deployment, scaling, and management is necessary to deploy a containerized application in a large-scale manner [11]. Kubernetes (K8s) [12], Docker Swarm [9], Red Hat OpenShift [13], or Amazon Elastic Container Service (ECS) [14] are the most used orchestration platforms and provide the necessary infrastructure to manage the life cycle of containers.

Scalability has been a general issue in SDN controllers [15]. Monolithic SDN controllers are deployed as a single unit, and the only way they can scale is in a vertical manner, i.e., adding more resources to the server the application is running in. They can manage medium-size networks, but the centralization can make the SDN controller the bottleneck on high flow request environments like data centers. As an example, the NOX [16] controller can handle up to 30,000 requests/s, which in large data centers can be insufficient, where the flow rate can reach 10 million requests/s [17]. A report about large-scale deployments can be found at [18]. In [19], the authors divided traditional SDN controllers by the scalability approach they offer into two main categories, topology- and mechanism-related approaches. In the topology-related approaches, the SDN controller tries to mitigate the scalability issues by distributing the controllers, in flat, hierarchical or hybrid designs, instead of having a unique SDN controller. The mechanism-related approaches are divided into parallelism-based, using

multithreaded programming to take advantage of multi-core CPUs, and routing optimizations. The approach followed in this paper falls into the mechanism-related approaches. Using microservices together with an orchestration platform, a cloud-native SDN controller is able to replicate the services of the controller that are causing the bottleneck into another server and redirect part of the requests to that replica, alleviating the high load of the original service, i.e., it is able to scale horizontally. This way, a cloud-native SDN controller can help solve the concern about scalability in centralized SDN architectures [20].

This paper extends the work presented in [21], where a cloud-native SDN controller with an ML framework and a deep neural network (DNN) model was introduced. Here, we provide a deeper description of the cloud-native SDN controller and extend it with two other optical quality of transmission (QoT) estimators based on ML, one based on the k-nearest neighbors (k-NN) and another based on support vector regression (SVR). The dataset used to train the three models was acquired from real-world data by means of a wavelength-division multiplexing (WDM) over SDM testbed, which is also made public. Moreover, a comparison between the three models is also provided in terms of performance. The scalability of the SDN controller has also been studied to assess the validity of the proposed architecture for large-scale deployments.

This paper is organized as follows. In Section 2 we evaluate the state of the art for cloud-native and ML for SDN controllers. Section 3 describes the architecture of the cloud-native SDN controller with ML capabilities and details the workflow for the composition of the internal topology, the creation of a connectivity service, and how the controller harvests telemetry from the chosen devices. Section 4 portrays the experimental scenario used to evaluate the architecture and describes the dataset that was gathered and used. Then, Section 5 presents the results. Finally, Section 6 concludes the paper.

## 2. RELATED WORK

QoT estimation in optical networks has been the focus of ML algorithms in the past few years, meant to improve their accuracy or speed [22]. Nonetheless, the speed of ML estimators depends on the algorithm, and they can be very computationally expensive. Another major problem is the source of the data used to train the models, as having real instead of synthetic data is hard to obtain.

Since the cloud-native approach has been proposed as the next step for SDN controllers, it has been discussed and preliminarily tested in a few articles and research projects. In [20,23], the authors experimentally showed a basic cloud-native SDN controller and studied the latency overhead that the communication between microservices introduced, and they performed basic demonstrations of two of the most fundamental benefits of the cloud-native approach, self-healing and auto-scaling. In [24,25] the authors demonstrated the ability of a microservices-based SDN controller to swap modules on the fly, using Open Network Operating System [26] (ONOS) or OpenDaylight [27] (ODL) SDN controllers (monolithic) as the core module. Other projects hosted by

the Linux Foundation [28] that started as monolithic controllers have changed their path and have begun the transition to a microservices-based architecture.  $\mu$ ONOS [29] is the cloud-native architecture redesign of ONOS, it is based on microservices, but it is still very experimental. ODL-Micro is another project based on ODL, which aims to decouple its components into microservices.

In [30], a telemetry-enabled cloud-native SDN controller was deployed on a real testbed and proved that horizontal scaling is necessary when dealing with a cloud-level number of transponders on an optical transport network. In [31], they showed the scalability ability using another QoT predictor based on the GN model, GNPY [32], using the same cloud-native architecture. In [33], the authors made use of transfer learning on a multi-domain EON (MD-EON) to improve the scalability on the learning process of an ML QoT predictor.

To the best of our knowledge, this is the first work to propose an ML-based QoT prediction mechanism that relies on horizontal scaling for (logically) centralized cloud-native SDN controllers.

### 3. CLOUD-NATIVE SDN CONTROLLER ARCHITECTURE

As introduced in Section 1, a cloud-native application has its modules developed as microservices, in which we differentiate four types: HTTP, gRPC, database, and scalable microservices. They are loosely coupled services, developed independently, in which communication between them is done by means of lightweight protocols to reduce overhead. The use of gRPC is very common in microservice architectures, as the use of HTTP/2 supports multiplexing of requests and responses as well as making the use of binary headers instead of text as HTTP 1.1 does. It also uses binary data instead of JSON as payload, making the protocol more efficient in terms of bandwidth [7]. The proposed cloud-native SDN controller ( $\mu$ ABNO) has a design based on the IETF Application-Based Network Operations (ABNO) [34] and has its modules depicted on Fig. 1.

Four different kinds of microservices are described: HTTP, gRPC, database, and scalable microservices. The HTTP microservices are used to interact with the user. The gRPC microservices are general purpose microservices that perform their operations and communicate with other microservices. The database microservices are used to keep and serve the data needed by other microservices. Finally, the scalable microservices perform tasks with high computational cost and can scale horizontally if needed to maintain the delay of the operation within an acceptable range, based on standard (CPU or RAM), or custom metrics (e.g., connections per second or delay of operations).

A detailed description of the modules is given:

- **NBI:** Manages the incoming HTTP requests from the user and hands them to the appropriate module responsible for the main workflow of the requested operation.
- **Connectivity:** Manages the workflows for the connectivity services. This module is where most of the intelligence of the SDN controller resides. It needs to connect to most

of the other modules to provision the connections to each domain, get the telemetry from the devices, configure the optical transponders, or compute a path.

- **Context:** Manages the database of the controller, i.e., stores the information. It is used to read, write, delete, or update data about the topologies, such as nodes, ports, and links, as well as additional information that is technology dependent.
- **Path Computation:** Manages the resolution of an end-to-end path. It determines a suitable route between input and output ports based on the topology information given by the Context module.
- **ML Analytics QoT Model (MLAQM):** Manages the ML QoT models that are going to be used by the MLAQP module. This module is constantly fed with data coming from the telemetry module. It updates the model it serves after a determined number of new samples are received. Then it saves the updated model to be ready to share it with the MLAQP.
- **ML Analytics QoT Predictor (MLAQP):** Resolves the QoT prediction requests made by other modules using ML models. It gets prediction requests made by other modules, gets an ML model stored by the MLAQM, and predicts the requested data. It is the scalable microservice demonstrated in this architecture. As it is a module that can consume a lot of computational resources, it scales horizontally when the resources reach a certain limit, deploying new replicas of the module to help with the requests.
- **Connection:** Manages the requests going to the underlying SDN controllers; i.e., it acts as the SBI module. It is developed with a plug-in architecture to be able to support different protocols, such as TAPI, OpenConfig, OpenROADM, etc.
- **Topology:** Manages the underlying topologies: retrieves the topologies from external devices and translates them to the internal data models. It is also developed with a plug-in architecture.
- **Telemetry:** Manages the retrieval of the telemetry data for the different configured devices. It is used to feed data to the MLAQM module by pulling data from the devices. This module enables an incremental learning approach, constantly gathering new data that can be fed into the MLAQM module so it can update the models when it receives the new data. It is also developed with a plug-in architecture.
- **Transponder:** Manages the configuration of the optical transponders, i.e., acts as the SBI for the optical transponders. It is also developed with a plug-in architecture.

These microservices deployed as containers are orchestrated by means of a cloud controller, which is responsible for the management of the life cycle of the containers, easing the operations of deploying, healing, networking, and scaling. The most important feature of the orchestrators for the QoT estimators is the scaling capability. The orchestrator can detect when a service is overloaded and replicate that single service, instead of having to replicate the whole application. This is done by fetching the desired metric to be watched from the resource metrics' API of each replica. Then, the orchestrator calculates the average of the metric between all replicas and computes the following formula:

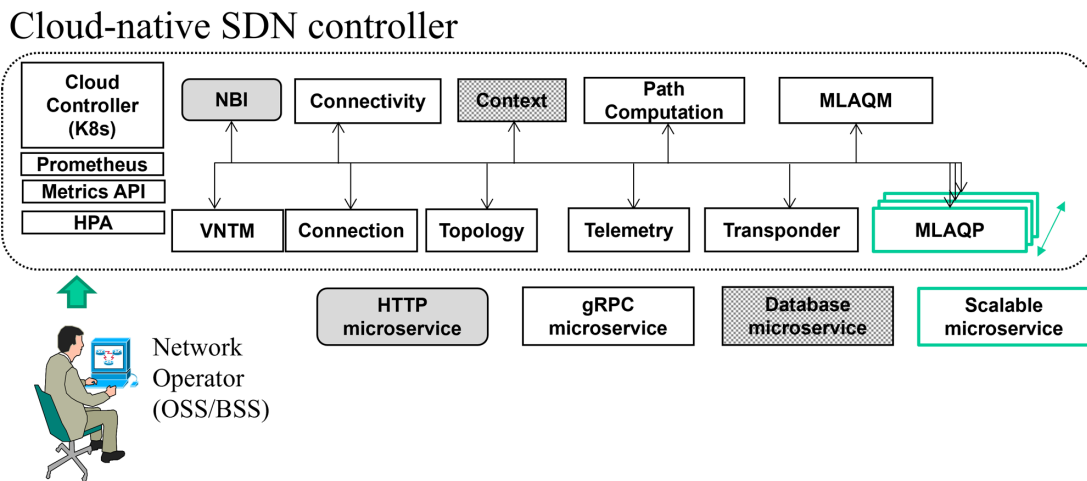


Fig. 1. SDN controller architecture.

$$\text{desiredReplicas} = \text{ceil} \left[ \text{currentReplicas} * \frac{\text{currentMetricValue}}{\text{desiredMetricValue}} \right].$$

This is done by retrieving the defined metrics to be watched each defined period of time and comparing them to the scaling threshold. This way, if the container is receiving more load than it is able to manage, more replicas are deployed on other servers, balancing the load between them and keeping a manageable load in each container.

#### A. Machine Learning QoT Model/Predictor Services

In ML techniques, first, it is needed to train the ML model so it is able to estimate or classify the new samples. Depending on how they treat the training samples, ML techniques can be divided into four different categories:

- Supervised learning: Algorithms in this category need labeled datasets to map to the output values or classes.
- Unsupervised learning: Algorithms in this category do not need training datasets. It is only valid for classification purposes.
- Semi-supervised learning: It is a mix between the two former categories, where both labeled and unlabeled data is used.
- Reinforcement learning: This technique is used to learn policies by letting an agent learn in a specific environment using some defined actions in order to learn how to maximize some long-term reward.

All three QoT predictors considered in this paper fall into the supervised category: k-NN, artificial neural networks (ANNs), and SVR.

The k-NN [35] is an algorithm where the output value is the average of the closest k-nearest neighbors to the input sample. The parameters that can be used to tune the algorithm are the  $k$  value, which is the number of closest labeled samples to the input sample that is needed to calculate the average from, and the way the distance is measured (e.g., Euclidean, or Mahalanobis distances).

ANNs are computing systems whose concept is based on biological neural networks. ANNs can also be unsupervised. They consist of a certain number of simple processing nodes,

called units or neurons, organized in layers that use nonlinear activation functions (e.g., sigmoid or hyperbolic tangent functions) to transform their input signal into the output of the neuron. The output of each neuron is connected to the neurons of the next layer (just one or all of them) by variable link weights plus a bias term. An ANN has to have at least three layers: the input layer that receives the input vector, one or more hidden layers, and the output layer. There are different types of ANN depending on how their layers are connected (feedforward or recurrent networks) or the density of the connections (fully or sparsely connected). Deep deural networks (DNNs) [36] are a kind of ANN that feature more than one hidden layer, making them able to learn more complex functions than shallow (just one hidden layer) neural networks. The training of the neural network consists of tuning the link weights by means of a class of algorithms called backpropagation. Generally, it consists of calculating the gradient of a loss function for the weights of each of the links of the network and then updating the weights in a determined rate, called the learning rate. If the learning rate is too low, the DNN could take too long to learn the function, and if it is too high, it could over-fit easily, i.e., find a local minimum (adjust the weights to the particular training set instead of making it more general).

The SVR [37] algorithms make use of different kernel functions to map the input vector into a higher dimensional space and find the hyperplane that fits the most input vectors. There are two basic parameters that can be adjusted. The  $C$  coefficient is used to tune how well fitted the model should be with respect to the input samples. The  $\epsilon$  parameter controls how many support vectors the model should have. More support vectors improve the accuracy, at the cost of the complexity of the model.

The selected ML QoT model is generated by the MLAQM in an online manner; e.g., the model is updated every time a new connectivity service is provisioned. The telemetry module is responsible for getting the optical performance monitoring (OPM) data and the present state of the connections to send it to the MLAQM to create or update the model. If the model used has not yet achieved an acceptable mean squared error

(MSE) value or has not been created yet, the SDN controller can choose a traditional (analytical or high margin) method to calculate the QoT until the MSE has reached an acceptable value (whatever is considered enough by the network operator) in subsequent connections.

## B. Connectivity Service Provisioning and QoT Estimation

The complete workflow for the creation of an end-to-end connection and the update of the ML QoT model is shown in Fig. 2. First, the user of the SDN controller sends a request for the creation of a connectivity service to the NBI (step 1) and passes it to the Connectivity module (step 2). The Connectivity module then requests the path to the Path Computation module (step 3), where it then asks the Context module for the topology context (steps 4 and 5). With the received topology, the Path Computation module resolves a path (step 6) and asks the MLAQP if the estimation for the BER for the computed path is below the desired threshold (steps 7 and 8). If it is not, the Path Computation module keeps computing paths until one of them is suitable and returns the response (step 9). After that, for each network domain the path traverses, the Connectivity module sends a request to the Connection module (step 10). It sends the connection to the network agent (step 11), which in turn returns the response to the Connection module (step 12) and then to the Connectivity module (step 13). The same approach is followed for the provisioning of the transponders, where the Connectivity module asks the Transponder module to configure the transponder agent (steps 14–17).

Finally, after the network and transponder agents (the software pieces that hide the internal details and protocols to external users) are configured, the Connectivity module sends the response to the NBI (step 18) and the NBI to the user (step 19). Nonetheless, a parallel process is launched by the Connectivity module between steps 17 and 18. In it, the Connectivity module sends a Remote Procedure Call (RPC) to the Telemetry module so it acknowledges there has been a new connectivity service created (step 20). Then, it gets the current state of the connectivity services created from the Context module (steps 21 and 22) and asks for the BER of the transponder to the Transponder module (steps 23–26). After that, the Telemetry module creates a new sample for the ML QoT models using the information about current connectivity services and the BER received from the Telemetry module (step 27) and sends an RPC to the MLAQM module with it to update the dataset (step 28). The MLAQM then stores the new sample, and if the number of new samples stored is bigger than a defined threshold, it updates the model (step 29) so the MLAQP can use a more accurate model in the future.

## 4. EXPERIMENTAL VALIDATION

### A. Experimental Scenario

Listing 1. HPA MLAQP configuration file.

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: mlanalyticspredictorservice
5  spec:
6    selector:
7      matchLabels:
8        app: mlanalyticspredictorservice
9    template:
10   metadata:
11     labels:
12       app: mlanalyticspredictorservice
13   spec:
14     terminationGracePeriodSeconds: 5
15     volumes:
16     - name: task-pv-storage
17       persistentVolumeClaim:
18         claimName: task-pv-claim
19     containers:
20     - name: server
21       image: myregistrydomain.com/mlanalyticspredictorservice
22       [...]
23     resources:
24       requests:
25         cpu: 500m
26         memory: 512Mi
27       limits:
28         cpu: 500m
29         memory: 2048Mi
30     imagePullSecrets:
31     - name: regcred
32
33 ---
34 apiVersion: autoscaling/v2beta1
35 kind: HorizontalPodAutoscaler
36 metadata:
37   name: mlanalyticspredictorservice
38 spec:
39   scaleTargetRef:
40     apiVersion: apps/v1
41     kind: Deployment
42     name: mlanalyticspredictorservice
43   minReplicas: 1
44   maxReplicas: 20
45   metrics:
46   - type: Resource
47     resource:
48       name: cpu
49       targetAverageUtilization: 35

```

The experimental setup is based on a cloud-native SDN controller deployed at CTTC in Barcelona (Spain) and data plane hardware and SDN agents deployed at KDDI Research in Saitama (Japan). The connection between these two premises was established through OpenVPN tunnels across the Internet.

The control plane consisted of the cloud-native SDN controller group of microservices deployed using the Docker container platform and the Kubernetes orchestration platform. Docker was responsible for the creation of the isolated containers for each module of the SDN controller, while Kubernetes was responsible for the automation of the deployment and scaling of such containers. The Horizontal Pod Autoscaler (HPA) of Kubernetes was responsible for the deployment of the replicas of the containers. The configuration of the shared disk space, available resources, and HPA for the MLAQP is shown on Listing 1. The container default resources were 50% of a core full capacity and 512 MiB, while the limits were the same 50% and 2048 MiB. The service started with one container and was set to a maximum of 20. It used the default 15 s as the time between pooling of the metrics to determine if there is a need for the replication of the container, in this case, the CPU utilization, which was set at 35% of a core capacity.

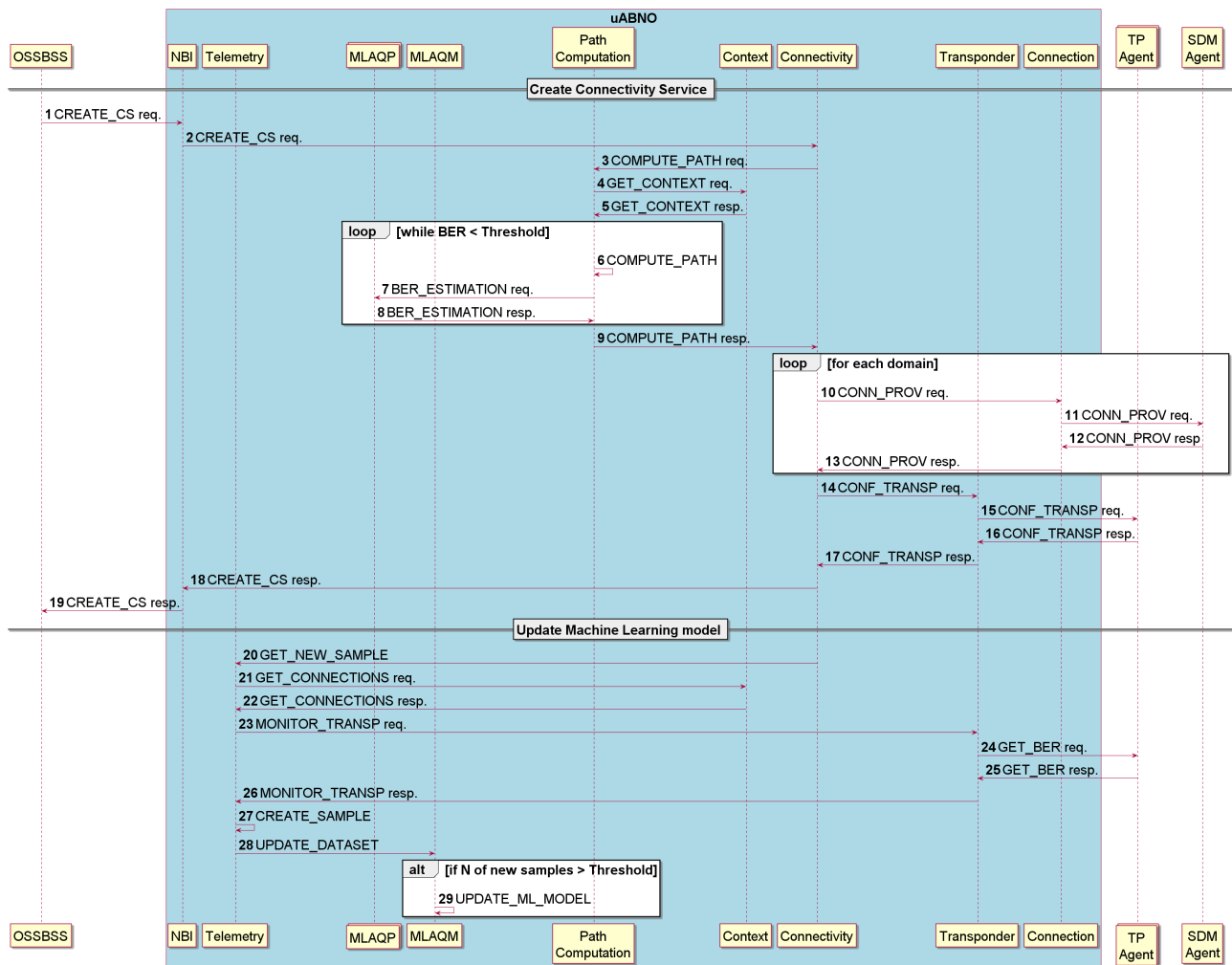


Fig. 2. Connectivity service provisioning workflow.

The connection module made use of a REST plugin to communicate with the SDN agent SDM, while the Transponder module communicated with the transponders by means of the NETCONF plugin, and the Telemetry plugin got the data from the transponders using the gRPC Network Management Interface (gNMI) plugin. The Topology module used a static configuration file to load the topology of the three domains, i.e., the Tx WDM domain, the SDM domain, and the Rx WDM domain.

The data plane hardware consisted of a WDM/SDM network domain, which comprised four transponders, two reconfigurable optical add-drop multiplexers (ROADMs), two optical switches, an 11 km SDM transmission line (i.e., 19-core fiber [38]) with a fan-in device, a fan-out device, and SDN agents. The four tunable transponders (ADVA FSP3000) operated on a frequency range from 193.2 to 193.5 THz following the 100 GHz ITU grid and were connected to the WDM/SDM domain. Each transponder was capable of two transmission modes, 100 and 200 Gb/s with modulation schemes of dual-polarization quadrature phase-shift keying (DP-QPSK) and 16 quadrature amplitude modulation (16QAM), respectively, and were controlled by the SDN controller via NETCONF. The ROADMs had multiple

inputs/outputs and supported a flexible grid in the C-band. Their inner configuration was based on wavelength-selective switches (WSSs) for multiplexing and demultiplexing of the outgoing and incoming optical signals and erbium-doped fiber amplifiers that were used as preamplifiers and boosters. The two optical switches that were deployed on each side of the 19-core fiber and were connected to the fan-in and fan-out devices were non-blocking all-optical matrix switches with an  $8 \times 8$  layout, with port #3 unavailable at the moment. Both the ROADMs and optical switches were controlled by their respective SDN controllers, and the way all the components were interconnected is depicted in Fig. 3.

On the transmitter's side, three of the transponders were connected to the ROADM, whose output ports were then connected to the input ports of the optical switch, whereas the fourth transponder was directly connected to an input port of the same optical switch. The ROADM provided a WSS functionality where the lightpaths of the three transponders could be independently switched to any of the used outgoing ports as long as the lightpaths were tuned on different wavelengths. The optical switch, on the other hand, would switch the optical signals from any ingoing port to any outgoing port with the limitation that signals from different input ports could not

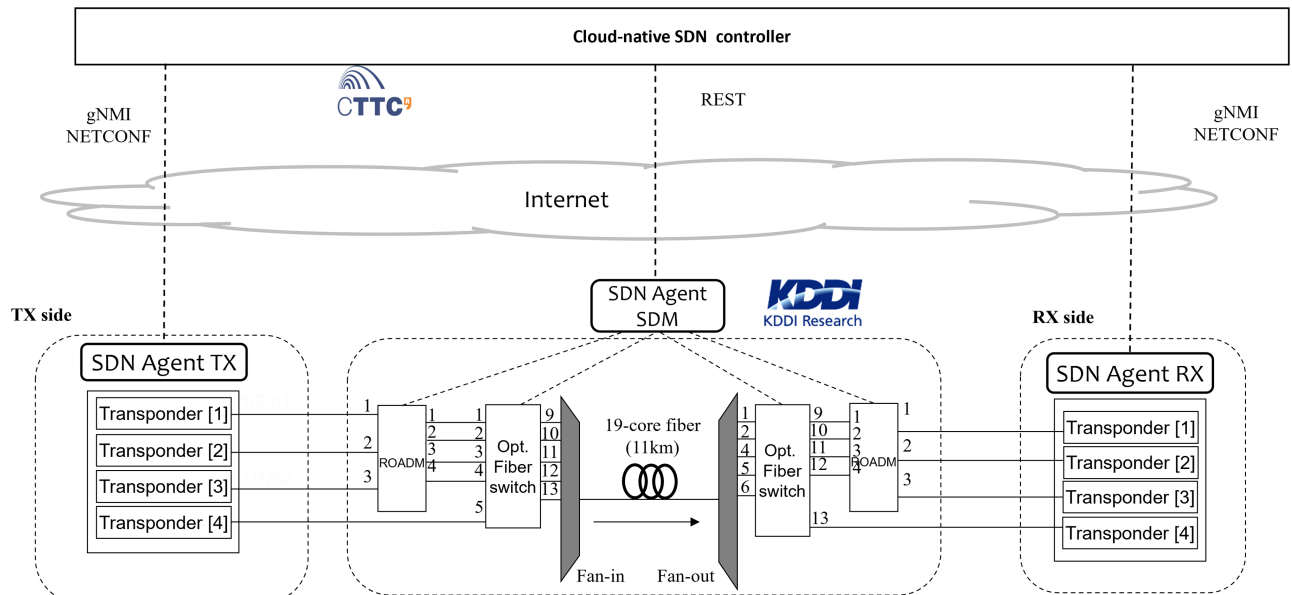


Fig. 3. Experimental scenario.

be forwarded to the same outgoing port. The output ports of the optical switch were then connected to the fan-in device in order to forward the optical signals into the SDM transmission line. Thus, based on the hardware limitations, the three lightpaths that went through the ROADM would use different wavelengths but could be switched on the same or different cores of the SDM transmission line, whereas the fourth lightpath would always end up on a separate core and could use any of the available wavelengths. Thus, the ROADM and optical switch setup would give the option of creating the worst case of nonlinear interference, where the lightpaths that copropagate the same core use contiguous wavelengths, and worst-case inter-core crosstalk, where the fourth lightpath is configured to propagate an adjacent core and at the same wavelength as one of the other three lightpaths. After the SDM fiber transmission, the output ports of the fan-out device were connected to the optical switch, where the three lightpaths would be switched to one of the connected ports of the ROADM and then to the transponders, whereas the fourth lightpath would bypass the ROADM's ports and end up at its respective transponder at the receiver's side. The data plane, although it is a limited testbed, will allow us to detect real physical impairments, such as inter-core interference, and show the control plane being used on real hardware. Moreover, scalability on a real optical network would be much more needed since ML algorithms would need more computer capacity. The control plane, on the other side, which is the contribution of the paper, is not affected by the limitations on the testbed and could be scaled to a real optical network.

**B. Obtained Dataset**

The dataset, shown in Fig. 4, that was obtained from the experimental scenario consists of 4384 samples obtained by provisioning one to four end-to-end lightpaths attending the limitations described in Section 4.A and measuring the

Timestamp	BER	Frequency (GHz)	Path #1	Path #2	Path #3	Path #4
1608288361	0.002698	193500	3	5	0	6
1608288706	0.000445	193300	7	0	4	3
1608289045	0.001991	193500	3	6	0	5
1608289383	0.000874	193200	6	0	6	7

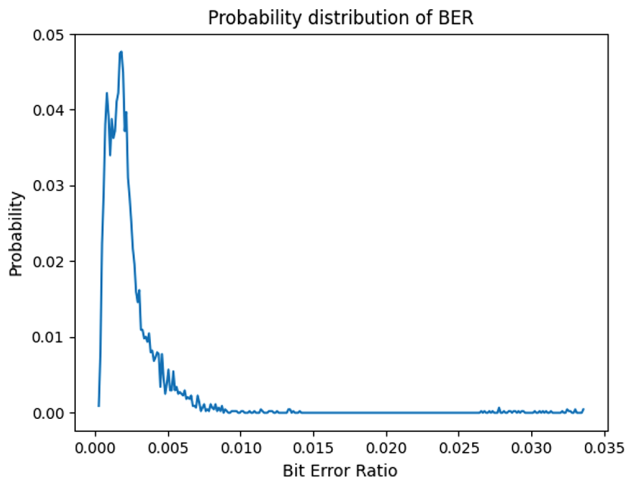
Fig. 4. First four samples of the dataset.

BER from transponder #1. The BER distribution can be seen in Fig. 5 with a standard deviation of 0.002760. Most samples are in the range of 0–0.015, but some can be seen in the range 0.02–0.033, which corresponds to the situation where transponder #1 is using core #5 or #6 and transponder #3 is using core #7, although not all of these situations have such a high BER. Each sample contains the timestamp in UNIX format, the BER for transponder #1, the operating frequency of transponder #1, and the spatial path (i.e., the SDM core) of each of the four transponders. As previously exposed, the BER ranges from 0 to 1 (ratio of error bits), the frequency value ranges from 193,200 to 193,500 GHz in jumps of 100 GHz, and the path ranges from 3 to 7 (the number of the fiber core) or 0 (unused path). The CSV file containing the dataset can be obtained at [39].

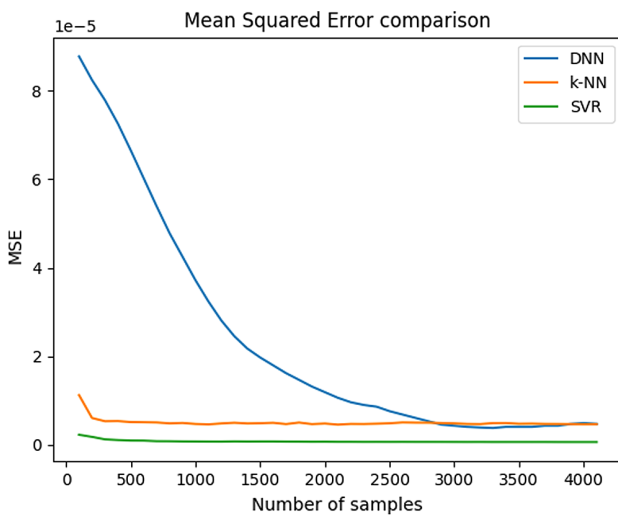
**C. QoT Models**

Three different ML models have been implemented to evaluate the scalability capabilities that they offer on this architecture: k-NN, SVR, and DNN, the first two using the scikit-learn [40] library and the latter using the Tensorflow [41] framework. They were trained with all the data from the dataset minus the timestamp, included in the dataset just as accessory information. All the columns but the BER were introduced as input data in one-hot coding.

The k-NN QoT estimator is calculated using 30 neighbors ( $n = 30$ ), and it uses the Euclidean distance as the metric for the distance.



**Fig. 5.** Probability distribution of the BER in the dataset.



**Fig. 6.** MSE of the DNN, k-NN, and SVR QoT estimators by number of samples.

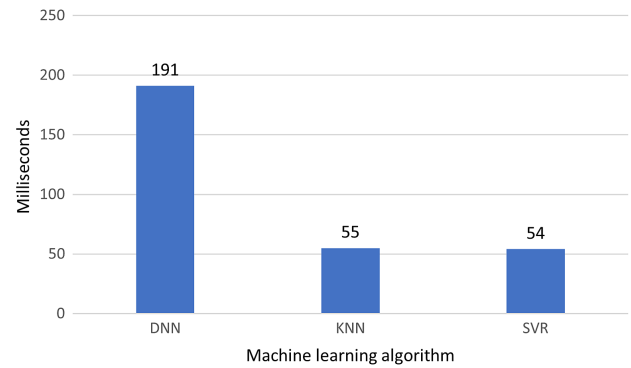
The DNN QoT estimator implemented consists of a feed-forward DNN with four fully connected layers, an input layer, two hidden layers, and the final output layer, with 48 neurons in total. Every neuron uses the rectified linear unit (ReLU) as the activation function, a learning rate of 0.000005, and the Adam algorithm as the optimizer.

Finally, the SVR algorithm uses the radial basis function (RBF) as the kernel function, with a C value of 1 and an epsilon value of 0.1.

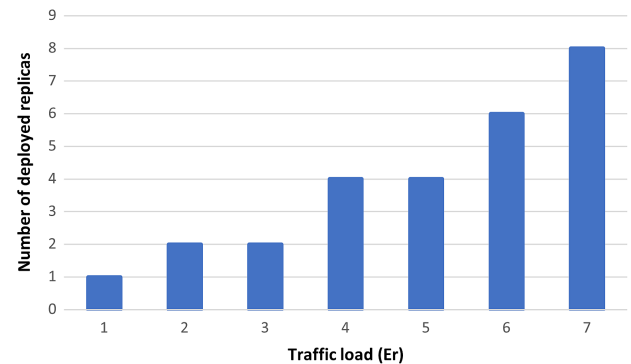
Figure 6 shows a performance comparison between the three models in terms of the MSE of the prediction, using a test split of the dataset, depending on the number of samples. It can be seen that the best performer is the SVR model followed by the k-NN and DNN, although the DNN model needs more samples to reach convergence.

## 5. RESULTS

The goal of the tests were to assess the cloud-native architecture for scaling ML QoT estimation. This was done by overloading



**Fig. 7.** Execution time of the three different ML QoT algorithms.



**Fig. 8.** Number of deployed replicas by requests per second with the different ML QoT algorithms.

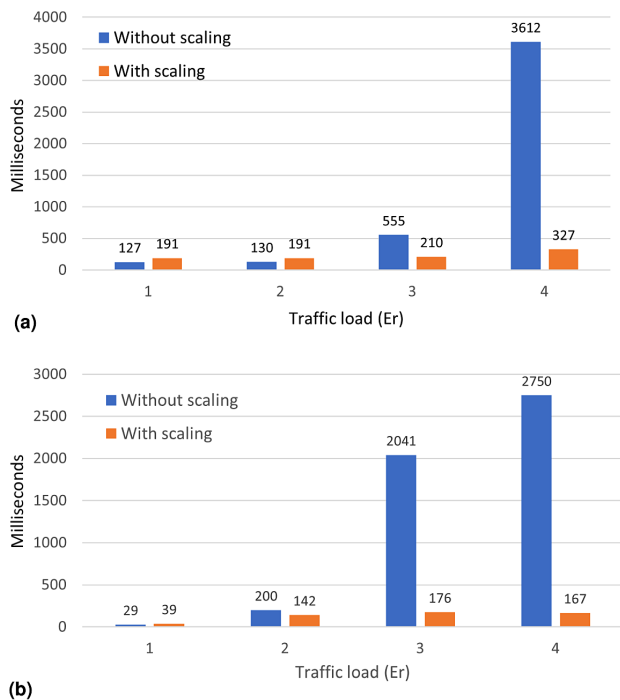
the controller requesting a large number of connectivity services per second that would render the MLAQP unable to serve all the requests for the BER predictor and force the container orchestrator to replicate the service. The tests were carried out on an AMD Ryzen Threadripper 3960 at 3.8 GHz with 32 GB of memory, running Ubuntu 20.04.2 LTS. The Docker version was 20.10.7, while the Kubernetes version was 1.21.3.

Figure 7 shows the execution time the three different algorithms take to infer a single sample. Both k-NN and SVR take almost the same time (55 and 54 ms, respectively), while the DNN takes about 3.5 times what both k-NN and SVR take (191 ms).

Figure 8 shows the number of replicas that are deployed over the traffic load (number of connectivity services requested per second) in erlangs (average simultaneous utilization of services) the SDN controller serves. It can be seen that the k-NN and SVR algorithms did not benefit from the scaling of the service, as the computing time is much faster. Meanwhile, the DNN model scales up to 8 replicas at 7 erlangs.

Figure 9(a) compares the average response time when 5000 connectivity services are requested to the SDN controller when the HPA is activated and when it is not. When the HPA is not activated, the average response time at 1 connectivity service per second is contained, but it goes up to more than 3.5 s when there are four requests per second. Likewise, when the HPA is activated, the response time increases with the erlangs requested. The reason why the DNN with the HPA activated performs slightly worse at 1 and 2 erlangs may be due to the periodical retrievals and computations on the metrics





**Fig. 9.** Behavior of the SDN controller using the DNN with increasing load. (a) Response time by number of requests per second. (b) Standard deviation by number of requests per second.

the system scales to, in this case the CPU usage. However, the response time stays below 327 ms even when the SDN controller handles 4 erlangs, more than an eleven-fold decrease in the response time.

In Fig. 9(b), the standard deviation on the same cases is shown. While in the non-HPA case the standard deviation of the response times grows in an exponential manner, when the HPA is activated, the standard deviation of the response times stays enclosed between 39 and 167 ms. No load higher than 4 erlangs could be considered on this test because it is the limit of the system load when no HPA is considered on the DNN model.

## 6. CONCLUSIONS

We have presented a cloud-native architecture for SDN controllers with a scalable ML QoT module. Three different models of ML QoT prediction were compared in terms of speed and performance within this architecture. It was shown how an ML QoT estimator based on DNNs is not capable of supporting high load scenarios and how it can be solved using horizontal scaling.

Future works could consider the scalability of the MLAQP module with dedicated ML hardware or implementing the scalability on other modules, where other high-intensity computation processes could happen, such as routing optimization or the control of the optical equipment.

**Funding.** Ministry of Internal Affairs and Communications (JPMI00316); European Commission (101015857); Ministerio de Economía, Industria y Competitividad, Gobierno de España (TI2018-099178-B-I00).

## REFERENCES

- N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.* **38**, 69–74 (2008).
- ONF Transport API (TAPI) [accessed 29 October 2021], <https://github.com/OpenNetworkingFoundation/TAPI>.
- J. Shao, X. Liang, and S. Kumar, "Comparison of split-step Fourier schemes for simulating fiber optic communication systems," *IEEE Photon. J.* **6**, 7200515 (2014).
- P. Poggiolini, G. Bosco, A. Carena, V. Curri, Y. Jiang, and F. Forghieri, "The GN-model of fiber non-linear propagation and its applications," *J. Lightwave Technol.* **32**, 694–721 (2013).
- D. Comer and A. Rastegarnia, "Toward disaggregating the SDN control plane," *IEEE Commun. Mag.* **57**(10), 70–75 (2019).
- P. Mell and T. Grance, "The NIST definition of cloud computing," Tech. Rep. 800-145 (National Institute of Standards and Technology (NIST), 2011).
- R. Vilalta, C. Manso, N. Yoshikane, R. Casellas, R. Martínez, T. Tsuritani, I. Morita, and R. Muñoz, "Experimental evaluation of control and monitoring protocols for optical SDN networks and equipment [Invited Tutorial]," *J. Opt. Commun. Netw.* **13**, D1–D12 (2021).
- C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: a state-of-the-art review," *IEEE Trans. Cloud Comput.* **7**, 677–692 (2017).
- Docker [accessed 29 October 2021], <https://www.docker.com>.
- Linux containers [accessed 29 October 2021], <https://linuxcontainers.org>.
- T.-T. Nguyen, Y.-J. Yeom, T. Kim, D.-H. Park, and S. Kim, "Horizontal pod autoscaling in Kubernetes for elastic container orchestration," *Sensors* **20**, 4621 (2020).
- Kubernetes [accessed 29 October 2021], <https://kubernetes.io>.
- Red Hat OpenShift [accessed 29 October 2021], <https://openshift.com>.
- Amazon elastic container service, [accessed 29 October 2021], <https://aws.amazon.com/ecs>.
- S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Commun. Mag.* **51**(2), 136–141 (2013).
- N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.* **38**, 105–110 (2008).
- D. Erickson, "The beacon openflow controller," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking* (2013), pp. 13–18.
- "ONOS Security and Performance Analysis (Report No. 2)," ONF Tech. Rep. (2018) [accessed 29 October 2021], [https://opennetworking.org/wp-content/uploads/2018/11/secperf\\_report\\_2.pdf](https://opennetworking.org/wp-content/uploads/2018/11/secperf_report_2.pdf).
- M. Karakus and A. Durresi, "A survey: control plane scalability issues and approaches in software-defined networking (SDN)," *Comput. Netw.* **112**, 279–293 (2017).
- R. Vilalta, J. L. de la Cruz, A. M. López-de-Lerma, V. López, V. L. Martínez, R. Casellas, and R. Muñoz, "uABNO: a cloud-native architecture for optical SDN controllers," in *Optical Fiber Communication Conference* (Optical Society of America, 2020), paper T3J.4.
- C. Manso, N. Yoshikane, R. Vilalta, R. Muñoz, R. Casellas, R. Martínez, C. Wang, F. Balasis, T. Tsuritani, and I. Morita, "First scalable machine learning based architecture for cloud-native transport SDN controller," in *Optical Fiber Communication Conference* (Optical Society of America, 2021).
- Y. Pointurier, "Machine learning techniques for quality of transmission estimation in optical networks," *J. Opt. Commun. Netw.* **13**, B60–B71 (2021).
- C. Manso, R. Vilalta, R. Casellas, R. Martínez, and R. Muñoz, "Cloud-native SDN controller based on micro-services for transport networks," in *6th IEEE Conference on Network Softwarization (NetSoft)* (IEEE, 2020), pp. 365–367.
- Q. P. Van, H. Tran-Quang, D. Verchere, P. Layec, H.-T. Thieu, and D. Zeglache, "Demonstration of container-based microservices

- SDN control platform for open optical networks,” in *Optical Fiber Communication Conference (OFC)* (Optical Society of America, 2019), paper M3Z.5.
25. Q. P. Van, D. Verchere, H. Tran-Quang, and D. Zeglache, “Container-based microservices SDN control plane for open disaggregated optical networks,” in *21st International Conference on Transparent Optical Networks (ICTON)* (IEEE, 2019).
  26. ONOS [accessed 29 October 2021], <http://www.onosproject.org>.
  27. OpenDaylight [accessed 29 October 2021], <https://www.opendaylight.org>.
  28. Linux Foundation [accessed 29 October 2021], <https://www.linuxfoundation.org>.
  29.  $\mu$ ONOS [accessed 29 October 2021], <https://github.com/onosproject>.
  30. R. Vilalta, C. Manso, N. Yoshikane, R. Muñoz, R. Casellas, R. Martínez, T. Tsuritani, and I. Morita, “Telemetry-enabled cloud-native transport SDN controller for real-time monitoring of optical transponders using GNMI,” in *European Conference on Optical Communications (ECOC)* (IEEE, 2020).
  31. C. Manso, R. Vilalta, R. Muñoz, R. Casellas, and R. Martínez, “Scalable for cloud-native transport SDN controller using GNPY and machine learning techniques for QOT estimation,” in *Optical Fiber Communication Conference (OFC)* (Optical Society of America, 2021), paper M1B.5.
  32. GNPY [accessed 29 October 2021], <https://gnpy.readthedocs.io>.
  33. C.-Y. Liu, X. Chen, R. Proietti, and S. B. Yoo, “Performance studies of evolutionary transfer learning for end-to-end QoT estimation in multi-domain optical networks [Invited],” *J. Opt. Commun. Netw.* **13**, B1–B11 (2021).
  34. D. King, A. Farrel, Q. Zhao, V. Lopez, R. Casellas, Y. Kamite, Y. Tanaka, Y. Lee, and Y. Yang, “A PCE-based architecture for application-based network operations,” IETF RFC 7491 (2015).
  35. J. H. Friedman, F. Baskett, and L. J. Shustek, “An algorithm for finding nearest neighbors,” *IEEE Trans. Comput. C-24*, 1000–1006 (1975).
  36. W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, “A survey of deep neural network architectures and their applications,” *Neurocomputing* **234**, 11–26 (2017).
  37. M. Awad and R. Khanna, “Support vector regression,” in *Efficient Learning Machines* (Springer, 2015), pp. 67–80.
  38. D. Soma, Y. Wakayama, S. Beppu, S. Sumita, T. Tsuritani, T. Hayashi, T. Nagashima, M. Suzuki, H. Takahashi, K. Igarashi, I. Morita, and M. Suzuki, “10.16 peta-bit/s dense SDM/WDM transmission over low-DMD 6-mode 19-core fibre across C + L band,” in *European Conference on Optical Communication (ECOC)* (IEEE, 2017).
  39. C. Manso and R. Vilalta, “ML SDM dataset,” GitHub (2021) [accessed 29 October 2021] <https://github.com/CTTC-ONS/ml-sdm-dataset>.
  40. scikit-learn [accessed 29 October 2021], <https://scikit-learn.org/>.
  41. TensorFlow [accessed 29 October 2021], <https://www.tensorflow.org/>.