Knowledge-Defined Networking

Albert Mestres[1], Alberto Rodriguez-Natal[1], Josep Carner[1], Pere Barlet-Ros[1,2], Eduard Alarcón[1], Marc Solé[3], Victor Muntés-Mulero[3], David Meyer[4], Sharon Barkai[5], Mike J Hibbett[6], Giovani Estrada[6], Khaldun Ma'ruf[7], Florin Coras[8], Vina Ermagan[8], Hugo Latapie[8], Chris Cassar[8], John Evans[8], Fabio Maino[8], Jean Walrand[9] and Albert Cabellos[1]

[1] Universitat Politècnica de Catalunya, [2] Talaia Networks, [3] CA Technologies, [4] Brocade Communication, [5] Hewlett Packard Enterprise, [6] Intel R&D, [7] NTT Communications, [8] Cisco Systems, [9] University of California, Berkeley

ABSTRACT

The research community has considered in the past the application of Artificial Intelligence (AI) techniques to control and operate networks. A notable example is the Knowledge Plane proposed by D.Clark et al. However, such techniques have not been extensively prototyped or deployed in the field yet. In this paper, we explore the reasons for the lack of adoption and posit that the rise of two recent paradigms: Software-Defined Networking (SDN) and Network Analytics (NA), will facilitate the adoption of AI techniques in the context of network operation and control. We describe a new paradigm that accommodates and exploits SDN, NA and AI, and provide use-cases that illustrate its applicability and benefits. We also present simple experimental results that support, for some relevant use-cases, its feasibility. We refer to this new paradigm as Knowledge-Defined Networking (KDN).

CCS Concepts

•**Networks** → **Network design principles;** *Network services; Network performance modeling;*

Keywords

Knowledge Plane, SDN, Network Analytics, Machine Learning, NFV, Knowledge-Defined Networking

1. INTRODUCTION

D. Clark et al. proposed "A Knowledge Plane for the Internet" [1], a new construct that relies on Machine Learning (ML) and cognitive techniques to operate the network. A Knowledge Plane (KP) would bring many advantages to networking, such as automation (recognize-act) and recommendation (recognize-explain-suggest), and it has the potential to represent a paradigm shift on the way we operate, optimize and troubleshoot data networks. However, at the time of this writing, we are yet to see the KP prototyped or deployed. Why?

One of the biggest challenges when applying ML for network operation and control is that networks are inherently distributed systems, where each node (i.e., switch, router) has only a partial view and control over the complete system. Learning from nodes that can only view and act over a small portion of the system is very complex, particularly if the end goal is to exercise control beyond the local domain. The emerging trend towards logical centralization of control will ease the complexity of learning in an inherently distributed environment. In particular, the Software-Defined Networking (SDN) paradigm [2] decouples control from the data plane and provides a logically centralized control plane, i.e., a logical single point in the network with knowledge of the whole.

In addition to the "softwarization" of the network, current network data plane elements, such as routers and switches, are equipped with improved computing and storage capabilities. This has enabled a new breed of network monitoring techniques, commonly referred to as network telemetry [3]. Such techniques provide real-time packet and flow-granularity information, as well as configuration and network state monitoring data, to a centralized Network Analytics (NA) platform. In this context, telemetry and analytics technologies provide a richer view of the network compared to what was possible with conventional network management approaches.

In this paper, we advocate that the centralized control offered by SDN, combined with a rich centralized view of the network provided by network analytics, enable the deployment of the KP concept proposed in [1]. In this context, the KP can use various ML approaches, such as Deep Learning (DL) techniques, to gather knowledge about the network, and exploit that knowledge to control the network using logically centralized control capabilities provided by SDN. We refer to the paradigm resulting from combining SDN, telemetry, Network Analytics, and the Knowledge Plane as Knowledge-Defined Networking.

This paper first describes the Knowledge-Defined Networking (KDN) paradigm and how it operates. Then, it describes a set of relevant use-cases that show the applicability of such paradigm to networking and the benefits associated with using ML. In addition, for some use-cases, we also provide early experimental results that show their feasibility. We conclude the paper by analyzing the open research challenges associated with the KDN paradigm.

2. A KNOWLEDGE PLANE FOR SDN ARCHITECTURES

This paper restates the concept of Knowledge Plane (KP) as defined by D. Clark et al. [1] in the context of SDN architectures. The addition of a KP to the traditional three planes of the SDN paradigm results in what we call Knowledge-Defined Networking.

The *Data Plane* is responsible for storing, forwarding and

processing data packets. In SDN networks, data plane elements are typically network devices composed of line-rate programmable forwarding hardware. They operate unaware of the rest of the network and rely on the other planes to populate their forwarding tables and update their configuration.

The *Control Plane* exchanges operational state in order to update the data plane matching and processing rules. In an SDN network, this role is assigned to the –logically centralized– SDN controller that programs SDN data plane forwarding elements via a southbound interface. While the data plane operates at packet time scales, the control plane is slower and typically operates at flow time scales.

The *Management Plane* ensures the correct operation and performance of the network in the long term. It defines the network topology and handles the provision and configuration of network devices. In SDN this is usually handled by the SDN controller as well. The management plane is also responsible for monitoring the network to provide critical network analytics. To this end, it collects telemetry information from the control and data plane while keeping a historical record of the network state and events. The management plane is orthogonal to the control and data planes, and typically operates at larger time-scales.

The *Knowledge Plane*, as originally proposed by Clark, is redefined in this paper under the terms of SDN as follows: *the heart of the knowledge plane is its ability to integrate behavioral models and reasoning processes oriented to decision making into an SDN network*. In the KDN paradigm, the KP takes advantage of the control and management planes to obtain a rich view and control over the network. It is responsible for learning the behavior of the network and, in some cases, automatically operate the network accordingly. Fundamentally, the KP processes the network analytics collected by the management plane, either preprocessed data or raw data, transforms them into knowledge via ML, and uses that knowledge to make decisions (either automatically or through human intervention). While parsing the information and learning from it is typically a slow off-line process, using such knowledge automatically can be done at a time-scales close to those of the control and management planes. However, the trend is towards on-line learning for applications such as those described in section 4.

3. KNOWLEDGE-DEFINED NETWORKING

The Knowledge-Defined Networking (KDN) paradigm operates by means of a control loop to provide automation, recommendation, optimization, validation and estimation. Conceptually, the KDN paradigm borrows many ideas from other areas, notably from black-box optimization [4], neural-networks in feedback control systems [5], reinforcement learning [6] and autonomic self-* architectures [7]. In addition, recent initiatives share the same vision stated in this paper[1] [8], [9]. Fig. 2 shows the basic steps of the main KDN control. In what follows we describe these steps in detail.

Forwarding Elements & SDN Controller → Analytics Platform.

The *Analytics Platform* aims to gather enough information to offer a complete view of the network. To that end, it monitors the data plane elements in real time while they forward packets in order to access fine-grained traffic information. In addition, it queries the SDN controller to obtain control and management state. The analytics platform relies on protocols, such as NETCONF (RFC 6241), NetFlow (RFC 3954) and IPFIX (RFC 7011), to obtain configuration information, operational state and traffic data from the network. The most relevant data collected by the analytics platform is summarized below.

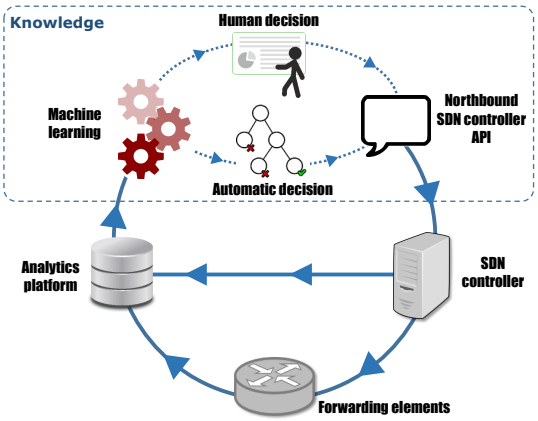

Figure 1: KDN operational loop

- Packet-level and flow-level data: This includes Deep Packet Inspection (DPI) information, flow granularity data and relevant traffic features.
- Network state: This includes the physical, topological and logical configuration of the network.
- Control & management state: This includes all the information included both in the SDN controller and management infrastructure, including policy, virtual topologies, application-related information, etc.
- Service-level telemetry: This is relevant to learn the behavior of the application or service, and its relation with the network performance, load and configuration.
- External information: This is relevant to model the impact of external events, such as activity on social networks (e.g., amount of people attending a sports event), weather forecasts, etc. on the network.

In order to effectively learn the network behavior, besides having a rich view of the network, it is critical to observe as many different situations as possible. As we discuss in Section 5, this includes different loads, configurations and services. To that end, the analytics platform keeps a historical record of the collected data.

Analytics Platform → Machine Learning.

ML algorithms (such as Deep Learning techniques) are the heart of the KP, which are able to learn from the network behavior. The current and historical data provided by the analytics platform are used to feed learning algorithms that learn from the network and generate knowledge (e.g., a model of the network). We consider three approaches: supervised learning, unsupervised learning and reinforcement learning.

In **supervised learning**, the KP learns a model that describes the behavior of the network, i.e., a function that

[1]Cognet project: http://www.cognet.5g-ppp.eu/

Table 1: KDN applications

	Closed Loop	Open Loop
Supervised	Automation Optimization	Validation Estimation What-if analysis
Unsupervised	Improvement	Recommendation
Reinforcement	Automation Optimization	N/A

relates relevant network variables to the operation of the network (e.g., the performance of the network as a function of the traffic load and network configuration). It requires labeled training data and feature engineering to represent network data.

Unsupervised learning is a data-driven knowledge discovery approach that can automatically infer a function that describes the structure of the analyzed data or can highlight correlations in the data that the network operator may be unaware of. As an example, the KP may be able to discover how the local weather affects the link's utilization.

In the **reinforcement learning** approach, a software agent aims to discover which actions lead to an optimal configuration. As an example the network administrator can set a target policy, for instance the delay of a set of flows, then the agent acts on the SDN controller by changing the configuration and for each action receives a reward, which increases as the in-place policy gets closer to the target policy. Ultimately, the agent will learn the set of configuration updates (actions) that result in such target policy. Recently, deep reinforcement learning techniques have provided important breakthroughs in the AI field that are being applied in many network-related fields (e.g., [10]).

Please note that learning can also happen offline and applied online. In this context knowledge can be learned offline training a neural network with datasets of the behavior of a large set of networks, then the resulting model can be applied online.

Machine Learning → Northbound controller API.

The KP eases the transition between telemetry data collected by the analytics platform and control specific actions. Traditionally, a network operator had to examine the metrics collected from network measurements and make a decision on how to act on the network. In KDN, this process is partially offloaded to the KP, which is able to make -or recommend- control decisions taking advantage of ML techniques.

Depending on whether the network operator is involved or not in the decision making process, there are two different sets of applications for the KP. We next describe these potential applications and summarize them in table 1.

Closed loop: When using supervised or reinforcement learning, the network model obtained can be used first for *automation*, since the KP can make decisions automatically on behalf of the network operator. Second, it can be used for *optimization* of the existing network configuration, given that the learned network model can be explored through common optimization techniques to find (quasi)optimal configurations. In the case of unsupervised learning, the knowledge discovered can be used to automatically *improve* the network via the interface offered by the SDN controller. For instance the relation between traffic, routing, topology and the resulting delay can be modeled to then apply optimal routing configurations that minimize delay.

Open loop: In this case the network operator is still in charge of making the decisions, however it can rely on the KP to ease this task. When using supervised learning, the model learned by ML can be used for *validation* (e.g., to query the model before applying tentative changes to the system). The model can also be used as a tool for performance *estimation* and *what-if analysis*, since the operator can tune the variables considered in the model and obtain an assessment of the network performance. When using unsupervised learning, the correlations found in the explored data may serve to provide *recommendations* that the network operator can take into consideration when making decisions.

Northbound controller API → SDN controller.

The northbound controller API offers a common interface to, human, software-based network applications and policy makers to control the network elements. The API offered by the SDN controller can be either a traditional imperative language or a declarative one [11]. In the latter case, the users of the API express their intentions towards the network, which then are translated into specific control directives.

The KP can operate both on top of imperative or declarative languages as long as it is trained accordingly. However, and at the time of this writing, developing truly expressive and high-level declarative northbound APIs is an open research question. Such intent-based declarative languages provide automation and intelligence capabilities to the system. In this context, we advocate that the KP represents an opportunity to help on their development, rather than an additional level of intelligence. As a result, we envision the KP operating on top of imperative languages, while helping on the translation of the intentions stated by the policy makers into network directives.

SDN controller → Forwarding Elements.

The parsed control actions are pushed to the forwarding devices via the controller southbound protocols in order to program the data plane according to the decisions made at the KP.

4. USE-CASES

This section presents a set of specific uses-cases that illustrate the potential applications of the KDN paradigm and the benefits a KP based on ML may bring to common networking problems. For two representative use-cases, we also provide early experimental results that show the technical feasibility of the proposed paradigm. All the datasets used in this paper, as well as codes and relevant hyper-parameters, can be found at [12].

4.1 Routing in an Overlay Network

The main objective of this use-case is to show that it is possible to model the behavior of a network with the use of ML techniques. In particular, we present a simple proof-of-concept example in the context of overlay networks, where an Artificial Neural Network (ANN) is used to build a model of the delay of the (hidden) underlay network, which can

later be used to improve routing in the overlay network.

Overlay networks have become a common solution for deployments where one network (overlay) has to be instantiated on top of another (underlay). This may be the case when a physically distributed system needs to behave as a whole while relying on a transit network, for instance a company with geo-distributed branches that connects them through the Internet. Another case is when a network has to send traffic through another for which it is not interoperable, for example when trying to send Ethernet frames over an IP-only network.

In such cases, an overlay network can be instantiated by means of deploying overlay-enabler nodes at the edge of the transit network and then tunneling overlay traffic using an encapsulation protocol (e.g., LISP (RFC 6830), VXLAN (RFC 7348), etc.). In many overlay deployments, the underlay network belongs to a different administrative domain and thus its details (e.g., topology, configuration) are hidden to the overlay network administrator (see fig. 2 inset).

Typically, overlay edge nodes are connected to the underlay network via several links. Even though edge nodes have no control over the underlay routing, they can distribute the traffic among the different links they use to connect to it. Edge nodes can use overlay control plane protocols (e.g., LISP) to coordinate traffic balancing policies across links. However, a common problem is how to find best/optimum per-link policies at the edge such that the global performance is optimized. An efficient use of edge nodes links is critical since it is the only way the overlay operator can control –to a certain extent– the traffic path over the underlay network.

Overlay operators can rely on building a model of the underlay network to optimize the performance. However, building such a model poses two main challenges. First, neither the topology nor the configuration (e.g., routing policy) of the underlay network are known, and thus it is difficult to determine the path that each flow will follow. Second, mathematical or theoretical models may fall short to model such a complex scenario.

ML techniques offer a new tool to model hidden networks by analyzing the correlation of inputs and outputs in the system. In other words, ML techniques can model the hidden underlay network by means of observing how the output traffic behaves for a given input traffic (i.e., f(routing policy, traffic) = performance). For instance, if two edge node links share a transit node within the -hidden- underlay network, ML techniques can learn that the performance decreases when both of those links are used at the same time and therefore recommend traffic balancing policies that avoid using both links simultaneously.

4.1.1 Experimental Results

To assess the validity of this approach, we carried out the following simple experiment. We have simulated (Omnet++[2]) a network with 12 overlay nodes, 19 underlay elements and a total of 72 links. The simulation has the following traffic characteristics: overlay nodes randomly split the traffic independently of the destination node, the underlay network uses shortest path routing with constant link capacity, constant propagation delay and Poisson traffic generation. From the KP perspective, only the overlay nodes that send and receive traffic are seen, while the underlay network is hidden.

[2]https://omnetpp.org/

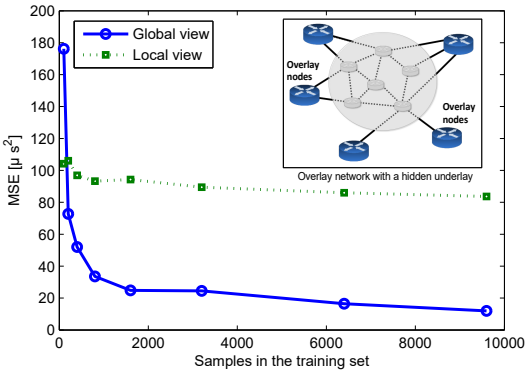

Figure 2: Prediction error (Mean Square Error) as a function of the size of the training set in an overlay-underlay scenario, both for global and local view.

We train an ANN using Pylearn2–Theano 0.7 with one hidden layer and a sigmoid activation function. The ANN is trained using the following input features: the traffic volume, defined as the aggregated bytes for the simulated time among source-destination pairs of the overlay nodes, and the routing of the overlay, defined as the ratio of traffic that is sent through each edge link. The average delays among paths obtained in the simulation are used as output features. We train the network with 9,600 training samples and we use 300 –separate samples– to validate the results.

With this use-case, we aim to learn the function that relates the traffic and the routing configuration of the overlay network with the resulting average delay of each path. That is, we train the ANN using the dataset that has as inputs traffic and routing configuration, and as output the average delay. Thus, the resulting ANN models the average delay of the packets for any traffic and routing configuration.

Fig. 2 shows the error (the accuracy) of the model as a function of the training set size (solid line). This error represents how accurately the model predicts the delay when the routing and traffic is known, but not the topology. As shown by the figure, the relative error is roughly 1% when using 6,400 training samples, equivalent to a mean square error of 20 μs^2. In addition to this, fig. 2 also shows (dashed line) when the model is trained only using local information. The main reason behind this experiment is that we aim to validate the main hypothesis stated in this paper: ML applied to a global view renders better results than when only local information is available. For this, each overlay node is trained only with local traffic, routing and delay and as the results show, the accuracy in this case is strongly degraded. This is because the delay between two nodes depends on the state of the queues of the underlay network, which in turns depends on the total traffic of the network.

Similar scenarios has been addressed before in the past (e.g., [13]) using network optimization techniques. Such mechanisms rely on models that represent the network built using either analytical techniques (e.g., Markov Chains) or computational models. In this paper we advocate that ML techniques represent a third pillar in network modeling, enabled by the global view and control offered by SDN and NA techniques. It provides important advantages: ML is data-driven, does not require simplifying assumptions typically found in traditional network modeling, works well with

complex systems (e.g., non-linearities and multi-dimensional dependencies) and, if trained well, can be general. Further information about this can be found at section 5.

4.2 Resource Management in an NFV scenario

This use-case shows how the KDN paradigm can also be useful in the context of Network Function Virtualization (NFV). NFV [14] is a networking paradigm where network functions (e.g., firewalls, load-balancers, etc.) no longer require specific hardware appliances but rather are implemented in the form of Virtual Network Functions (VNFs) that run on top of general purpose hardware.

The resource management in NFV scenarios is a complex problem since VNF placement may have an important impact on the overall system performance. The problem of optimal Virtual Machine (VM) placement has been widely studied for Data Center (DC) scenarios (see [15] and the references therein), where the network topology is mostly static. However, in NFV scenarios the placement of a VNF modifies the performance of the virtualized network. This increases the complexity of the optimal placement of VNFs in NFV deployments.

Contrary to the overlay case, in the VNF placement problem all the information is available, e.g., virtual network topology, CPU/memory usage, energy consumption, VNF implementation, traffic characteristics, current configuration, etc. However, in this case the challenge is not the lack of information but rather its complexity. The behavior of VNFs depend on many different factors and thus developing accurate models is challenging.

The KDN paradigm can address many of the challenges posed by the NFV resource-allocation problem. For example, the KP can characterize, via ML techniques, the behavior of a VNF as a function of the collected analytics, such as the traffic processed by the VNF or the configuration pushed by the controller. With this model, the resource requirements of a VNF can be modeled by the KP without having to modify the network. This is helpful to optimize the placement of this VNF and, therefore, to optimize the performance of the overall network.

4.2.1 *Experimental results*

To validate this use-case we model the CPU consumption of real-world VNFs when operating under real traffic. We have chosen two different network elements, an Open Virtual Switch (OVS v2.0.2[3]) and Snort (v2.9.6.0[4]). We have tested OVS with two different set of rules and controller configurations: as a SDN-enabled firewall and as a SDN-enabled switch. In both cases, we have aimed to have a representative configuration of real-world deployments. With this we have three different VNFs: SNORT, Firewall and Switch.

To measure the CPU consumption of both VNFs, we have deployed them in VMs (Ubuntu 14.04.1) running on top of a hypervisor (VMware ESXi v5.5), which provides a virtual network to interconnect the VNFs using 1 Gbps links. Two VMs generate and receive traffic, and are connected to the VNF. The traffic used in this experiment was replayed using tcpreplay (version 3.4.4) from an on-campus DPI infrastructure. The campus network serves around 30k users, further details about the traffic traces can be found in [16]. To represent the traffic, we extract off-line a set of 86 traffic features

[3]http://openvswitch.org/
[4]https://www.snort.org/

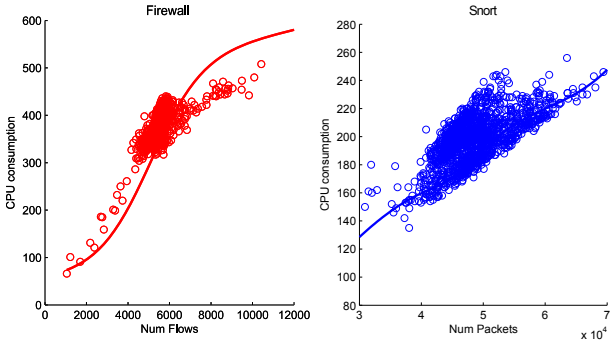

Figure 3: Measured points and the built model using two different features for two different VNF (only showing the most relevant feature)

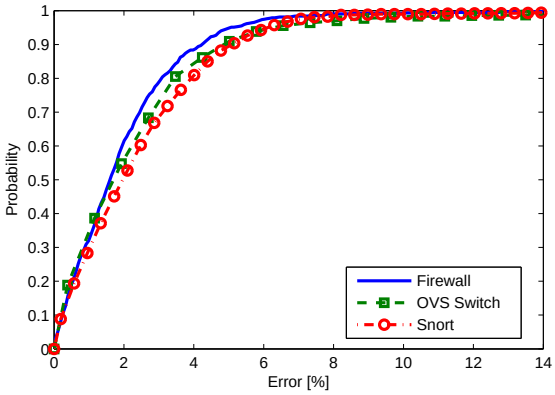

Figure 4: Cumulative Distribution Function of the relative error ($(y_{pred}-y_{real})/y_{real}$) for the three VNFs

in 20 second batches: number of packets, number of 5-tuple flows, average length, number of different IPs or ports, application layer information, among others. The complete set of features can be found in [12]. In the learning process, we use the Matlab ANN toolbox with one hidden layer, where the input are the 86 traffic features and the output is the measured CPU consumption. In this case, we aim to learn the function that relates the traffic features with the CPU consumption.

In this experiment, we use a dataset of 750 samples (600 for training and 150 for test-set) for the Firewall learning model and a dataset of 1,100 samples (900 for training and 200 for test-set) for the Snort and the Switch learning models. First we aim to understand if the model is complex (e.g., non-lineal) and thus, requires the use of ML. For this we show in fig. 3 the model when only a single feature is used, specifically we pick the input traffic feature that is more relevant to predict CPU of each of the VNFs, this is the result of a PCA analysis. The figure plots the predicted CPU consumption (line) and the measured data (dots) as a function of the traffic feature used for prediction. As the plot shows, training with a single feature leads to poor accuracy while showing non-linear dependencies, motivating the use of neural networks. Finally, fig. 4 shows the CDF of the relative error ($(y_{pred}-y_{real})/y_{real}$) of the models when trained with all the 86 features defined previously, achieving very good accuracy.

4.3 Knowledge extraction from network logs

Operators typically equip their networks with a logging infrastructure where network devices report events (e.g., link going down, packet losses, etc.). Such logs are extensively used by operators to monitor the health of the network and to troubleshoot issues. Log analysis is a well-known research field and, in the context of the KDN paradigm, it can also be used in networking [17]. By means of unsupervised learning techniques, a KDN architecture can correlate log events and discover new knowledge. This knowledge can be used by the network administrators for network operation using the open-loop approach, or to take automatic decisions in a closed-loop solution. These are some specific examples of Knowledge Discovery using Network Logging and unsupervised learning:

- Node N is always congested around 8pm and Services X and Y have an above-average number of clients.
- Abnormal number of BGP UPDATES messages sent and Interface 3 is flapping.
- Fan speeds increase in node N when interface Y fails.

4.4 5G mobile communications networks

The fifth generation of mobile communications networks will provide higher data rates, lower latencies, among other advances together with an important update of the network [18]. The 5G network is, by design, a Wireless SDN (WSDN), which offers a flexible network architecture, required by the specifications of 5G. Moreover, 5G is complemented by the use of Network Function Virtualization (NFV), to increase the flexibility of the 5G network and to create virtual networks over the same physical network. Within this context, KDN can be easily applied as in a conventional SDN+NFV network.

Additionally, 5G networks require novel technical solutions in which the KDN paradigm can also be helpful. The high scalability required makes necessary the design of intelligent routing algorithms for a large number of users, especially when these users are mobile [18]. The design of reliable handoffs, as well as the design of dynamic routing algorithms may take advantage of the data collected to predict the user movement to increase the performance of these algorithms. Moreover, this can also be used to increase the efficiency of the beam-steering techniques, which will facilitate the increase of the throughput in the physical layer.

5. CHALLENGES AND CONCLUSIONS

The KDN paradigm brings significant advantages to networking, but at the same time it also introduces important challenges that need to be addressed. In what follows we discuss the most relevant ones.

New ML mechanisms: Although ML techniques provide flexible tools to computer learning, its evolution is partially driven by existing ML applications (e.g., Computer Vision, recommendation systems, etc.). In this context the KDN paradigm represents a new application for ML and as such, requires either adapting existing ML mechanisms or developing new ones. Graphs are a notable example, they are used in networking to represent topologies, which determine the performance and features of a network. In this context, only preliminary attempts have been proposed in the literature to create sound ML algorithms able to model the topology of systems that can be represented through a graph [19]. Although such proposals are not tailored to network topologies, their core ideas are encouraging for the computer networks research area. In this sense, the combination of modern ML techniques, such as Q-learning techniques, convolutional neural networks and other deep learning techniques, may be essential to make a step further in this area.

Non-deterministic networks: Typically networks operate with deterministic protocols. In addition, common analytical models used in networking have an estimation accuracy and are based on assumptions that are well understood. In contrast, models produced by ML techniques do not provide such guarantees and are difficult to understand by humans. This also means that manual verification is usually impractical when using ML-derived models. Nevertheless, ML models work well when the training set is *representative* enough. Then, what is a *representative* training set in networking? This is an important research question that needs to be addressed. Basically, we need a deep understanding of the relationship between the accuracy of the ML models, the characteristics of the network, and the size of the training set. This might be challenging in this context as the KP may not observe all possible network conditions and configurations during its normal operation. As a result, in some use-cases a training phase that tests the network under various representative configurations can be required. In this scenario, it is necessary to analyze the characteristics of such loads and configurations in order to address questions such as: does the normal traffic variability occurring in networks produce a representative training set? Does ML require testing the network under a set of configurations that may render it unusable?

New skill set and mindset: The move from traditional networks to the SDN paradigm has created an important shift on the required expertise of networking engineers and researchers. The KDN paradigm further exacerbates this issue, as it requires a new set of skills in ML techniques or Artificial Intelligence tools.

Standardized Datasets: In many cases, progress in ML techniques heavily depends on the availability of standardized datasets. Such datasets are used to research, develop and benchmark new AI algorithms. And some researchers argue that the cultivation of high-quality training datasets is even more important that new algorithms, since focusing on the dataset rather than on the algorithm may be a more straightforward approach. The publication of datasets is already a common practice in several popular ML application, such as image recognition[5]. In this paper, we advocate that we need similar initiatives for the computer network AI field. For this reason, all datasets used in this paper are public and can be found at [12]. This datasets have proven useful in routing and VNF experiments, it is our hope that help kick-off a community contributing with larger datasets.

Summary: We advocate that in order to address such important challenges and achieve the vision shared in this paper, we require a truly inter-disciplinary effort between the research fields of Artificial Intelligence, Network Science and Computer Networks.

Acknowledgments. We would like to thank Prof. Shyam Parekh and David Kirsch for their valuable suggestions. We would also like to thank the anonymous reviewers for their

[5]Imagenet database: http://www.image-net.org/

comments that improved this paper. This work has been partially supported by the Spanish Ministry of Economy, Industry and Competitiveness and EU FEDER under grant TEC2014-59583-C2-2-R (SUNSET project) and by the Catalan Government (ref. 2014SGR-1427).

6. REFERENCES

[1] Clark, D., et al. "A knowledge plane for the Internet." *Conf. on Applications, technologies, architectures, and protocols for computer communications*, ACM, 2003.

[2] Kreutz, D., et al. "Software-defined networking: A comprehensive survey." *Proceedings of the IEEE*, vol. 103.1, pp. 14-76, 2015.

[3] Kim, C., et al. "In-band Network Telemetry via Programmable Dataplanes." *Industrial demo, ACM SIGCOMM*, 2015.

[4] Rios, L.M., and Sahinidis, N. V., "Derivative-free optimization: a review of algorithms and comparison of software implementations." *Journal of Global Optimization*, vol.54, pp. 1247-1293, 2013.

[5] Narendra, K. S., and Kannan P. "Identification and control of dynamical systems using neural networks." *Neural Networks, IEEE Trans.*, vol.1, pp. 4-27, 1990.

[6] Mnih, V., et al. "Human-level control through deep reinforcement learning." *Nature 518(7540)*, pp. 529-533, 2015.

[7] Derbel, H., et al. "ANEMA: Autonomic network management architecture to support self-configuration and self-optimization in IP networks." *Computer Networks*, vol. 53.3, pp. 418-430, 2009

[8] Zorzi, M., et at. "COBANETS: A new paradigm for cognitive communications systems." *International Conference on Computing, Networking and Communications (ICNC)*, pp. 1-7, 2016

[9] Giordano, Danilo, et al. "YouLighter: A cognitive approach to unveil YouTube CDN and changes." *IEEE Transactions on Cognitive Communications and Networking*, vol. 1.2, pp. 161-174, 2015.

[10] Mao, Hongzi, et al. "Resource Management with Deep Reinforcement Learning." *Hot Topics in Networks. ACM*, 2016.

[11] Foster, N., et al., "Languages for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 6, pp. 128-134, 2013.

[12] KDN database, http://knowledgedefinednetworking.org/

[13] Chen, Yan, et al. "An algebraic approach to practical and scalable overlay network monitoring." *ACM SIGCOMM Computer Communication Review*, vol. 34.4, 2004.

[14] Han, B., et al. "Network function virtualization: Challenges and opportunities for innovations." *Communications Magazine, IEEE*, vol. 53.2, pp. 90-97, 2015.

[15] Meng, X., et al. "Improving the scalability of data center networks with traffic-aware virtual machine placement." *INFOCOM, Proceedings IEEE*, 2010.

[16] Barlet-Ros, P., et at. "Predictive resource management of multiple monitoring applications." *Transactions on Networking, IEEE/ACM*, vol. 19(3), 2011.

[17] Kimura, Tatsuaki, et al. "Spatio-temporal factorization of log data for understanding network events." *IEEE INFOCOM*, pp. 610-618, 2014.

[18] Akyildiz, Ian, et al. "5G roadmap: 10 key enabling technologies." *Computer Networks*, vol 106, pp. 17-48, 2016.

[19] Bruna, J., et al. "Spectral networks and locally connected networks on graphs." *arXiv preprint* arXiv:1312.6203, 2013.