Part II

Management Models and Frameworks

3

Managing Virtualized Networks and Services with Machine Learning

Raouf Boutaba[1], Nashid Shahriar[2], Mohammad A. Salahuddin[1], and Noura Limam[1]

[1]David R. Cheriton School of Computer Science, University of Waterloo, Ontario, Canada
[2]Department of Computer Science, University of Regina, Saskatchewan, Canada

3.1 Introduction

Virtualization is instigating a revolutionary change in the networking industry, similar to that of the computer industry in the 1980s. Indeed, before IBM compatibles and Windows, the mainframe computer industry in the late 1970s and early 1980s was closed with vertically integrated specialized hardware, operating system and applications – all from the same vendor. A revolution happened when open interfaces started to appear, the industry became horizontal and innovation exploded. A similar revolution is happening in the networking industry, which previously had the "mainframe" mindset relying on vendor specific, proprietary and vertically integrated solutions. Network Virtualization (NV) and the provision of open interfaces for network programming are expected to foster innovation and rapid deployment of new network services.

The idea of NV gained momentum to address the Internet ossification problem by enabling radically different architectures [1]. The current Internet suffers from ossification, as the Internet size and rigidity make it difficult to adopt new networking technologies [2]. For example, the transition from Internet Protocol version 4 (IPv4) to IPv6 has started more than a decade ago, while IPv6 adoption rate is still significantly low as reported by major service providers (i.e. less than 30% of Google users have adopted IPv6 [3]). It is becoming increasingly cumbersome to keep up with emerging applications quality of service (QoS) requirements of bandwidth, reliability, throughput, and latency in an ossified Internet. NV solves the ossification problem by allowing the coexistence of multiple virtual networks (VNs), each customized for a specific purpose on the shared Internet. Although the

Communication Networks and Service Management in the Era of Artificial Intelligence and Machine Learning, First Edition. Edited by Nur Zincir-Heywood, Marco Mellia, and Yixin Diao.
© 2021 The Institute of Electrical and Electronics Engineers, Inc. Published 2021 by John Wiley & Sons, Inc.

idea of NV originated to address the Internet ossification, NV has been adopted as a diversifying attribute of different networking technologies, including wireless [4], radio access [5], optical [6], data center (DC) [7], cloud computing [8], service-oriented [9], software-defined networking (SDN) [10, 11], and Internet of Things (IoT) [12].

Another prolific application of virtualization in networking is the adoption of virtualized network services through network functions virtualization (NFV). NFV decouples network or service functions from underlying hardware, and implements them as software appliances, called virtual network functions (VNFs), on virtualized commodity hardware. Numerous state-of-the-art VNFs have already shown the potential to achieve near-hardware performance [13, 14]. Moreover, NFV provides ample opportunities for network optimization and cost reduction. First, hardware-based network or service functions come with high capital expenditures, which can be reduced by deploying VNFs on commodity servers. Second, hardware appliances are usually placed at fixed locations, whereas in NFV, a VNF can be deployed on any server in the network. VNF locations can be determined intelligently to meet dynamic traffic demand and better utilize network resources. NFV opens-up the opportunity to simultaneously optimize VNF locations and traffic routing paths, which can significantly reduce the network operational expenditure. Finally, hardware-based functions are difficult to scale, whereas NFV offers to cost-efficiently scale VNFs on-demand. A service-function chain (SFC) is an ordered sequence of VNFs composing a specific service [15]. For example in a typical DC network, traffic from a server passes through an intrusion detection system (IDS), a firewall, and a network address translator (NAT) before reaching the Internet.

Virtualizing networks and services facilitate a new business model, namely Network-as-a-Service (NaaS), which provides a separation between the applications and services, and the networks supporting them [16]. Network operators can adopt the NaaS model to partition their physical network resources into multiple VNs (also called *network slices*) and lease them to service providers [17]. In turn, service providers use VNs to offer services with diverse QoS requirements, without any investment in establishing and managing a physical infrastructure. A perfect incarnation of the NaaS model is network slicing for fifth generation (5G) mobile networks. Using network slicing, a single 5G physical network can be sliced into multiple isolated logical networks of varying sizes and structures, dedicated to different types of services. These “self-contained” VNs should be flexible enough to simultaneously accommodate diverse business-driven use cases from multiple service providers on a common network infrastructure, and created on-demand according to the service providers’ requirements.

The benefits of virtualized networks and services come at the cost of additional management challenges for network operators. First, a network operator has to orchestrate VNs/network slices in such a way that they can coexist in a single infrastructure, without affecting each other. Hence, smart orchestration decisions need to be carried out to provision VNs satisfying requirements from diverse users and applications, while ensuring desired resource utilization. This also involves configuring a large number of virtual instances and their operating parameters. The initial orchestration and configuration need to be adapted to cope with time-varying traffic demands and change in network states. Second, the added virtualization layer introduces new attack and failure surfaces across different administrative and technological domains. For instance, any failure in the underlying physical resource can propagate to the hosted virtual resources, though the reverse is not always true. Similarly, remediation and mitigation mechanism for one VN should not jeopardize the operation of coexisting VNs. These diverse challenges call for automated management that cannot be satisfied with the traditional, reactive human-in-the loop management approach. The management of VNs should be intelligent to leverage the sheer volume of operational data generated within a live network, and take automated decisions for different operational and management actions. Therefore, Artificial Intelligence (AI) and Machine Learning (ML) can play pivotal roles for realizing the automation of control and management for VNs and their services [18, 19].

AI and ML techniques have been widely used in addressing networking problems in the last few decades [18, 19]. However, when it comes to virtualized network management, the lack of real-world deployment of virtualized services impedes the application of AI and ML techniques. Despite that, there has been a recent surge in research efforts that aim to leverage ML in addressing complex problems in NV environment. This chapter summarizes state-of-the-art research and outlines potential avenues in the application of AI and ML techniques in virtualized network and service management. The rest of the chapter is organized as follows. We provide a detailed technology overview of virtualized networks and services in Section 3.2. We present state-of-the-art research that apply AI and ML in three core sub-areas of virtualized networks and services, namely NV, NFV, and network slicing in Section 3.3. We conclude the chapter in Section 3.4 with a brief summary, and outline possible research avenues to advance the state-of-the-art in applying AI and ML for managing virtualized networks and services.

3.2 Technology Overview

Virtualization in networking is not a new concept. Virtual channels in X.25-based telecommunication networks (e.g. ATM networks) allow multiple users to share a large physical channel. Virtual Local Area Networks (VLANs) partition a physical Local Area Network (LAN) among multiple logical LANs with elevated levels of trust, security, and isolation. Similarly, virtual private networks (VPNs) offer dedicated communications that connect multiple geographically distributed sites through private and secure tunnels over public communication networks (e.g. the Internet). Overlay networks (e.g. PlanetLab) create virtual topologies on top of the physical topology of another network. Overlays are typically implemented in the application layer, though various implementations at lower layers of the network stack do exist. These technologies deploy narrow fixes to specific problems without a holistic view of the interactions between coexisting VNs. Therefore, in this section, we provide a comprehensive overview of different technologies that enable virtualization of networks and services.

3.2.1 Virtualization of Network Functions

An Network Function (NF) is a functional block within a network infrastructure that has well-defined external interfaces and functional behavior [13]. NFs in traditional wired networks can be classified in two categories: forwarding functions and value-added functions. Forwarding functions, such as routers, switches, and transponders, provide the functionality to forward data along a network path. On the other hand, value-added functions, such as Dynamic Host Configuration Protocol (DHCP), Network Address Translation (NAT), Universal Plug and Play (UPnP), Firewall, Optimizer, and Deep Packet Inspectors (DPI), offer additional capabilities to the data forwarding path. Similarly, NFs in mobile networks are categorized in two classes: Radio Access Network (RAN) functions and core functions. We will discuss more about RAN and core functions later when we discuss network slicing. In this subsection, we discuss two popular methods of virtualizing NFs as follows (a summary is depicted in Figure 3.1).

3.2.1.1 Resource Partitioning

Partitioning is a convenient method to create multiple virtual entities on a single networking device (e.g. routers and switches) that provide forwarding functions. Resource partitioning can be achieved either by hard partitioning (i.e. dedicated switch ports, CPU cores, cards) or by soft partitioning (i.e. CPU execution capping, routing, and forwarding table partitioning). Hard partitioning provides excellent isolation, but it requires abundant hardware to implement. In contrast, soft partitioned instances may not provide the highest level of isolation and security due to their shared nature.

A hard partitioned router, called a Logical Router (LR), can run across processors on different cards of a router device. All the underlying hardware and software resources, including network processors, interfaces, and routing and forwarding tables, are dedicated to an LR. Examples of LR are "protected system domains" by Juniper Networks, or "logical routers" by Cisco Systems. Hardware partitioned

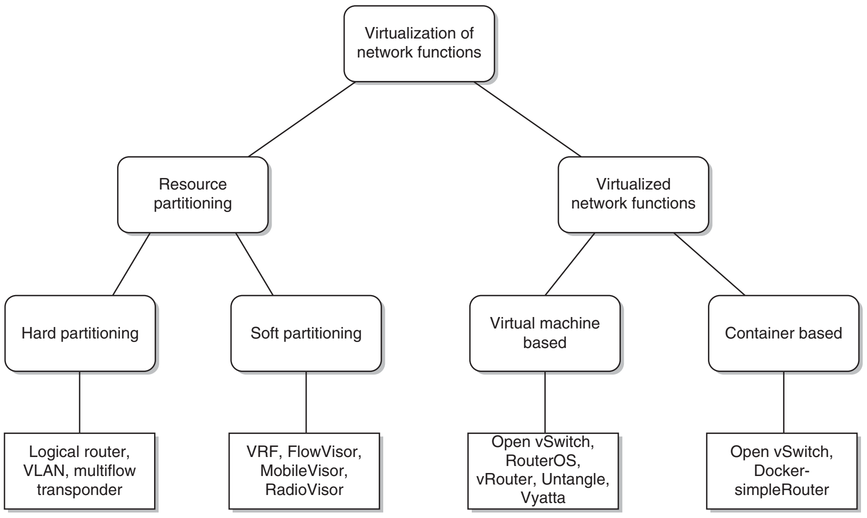
Virtualization of network functions
Resource partitioning
Virtualized network functions
Hard partitioning
Soft partitioning
Virtual machine based
Container based
Logical router, VLAN, multiflow transponder
VRF, FlowVisor, MobileVisor, RadioVisor
Open vSwitch, RouterOS, vRouter, Untangle, Vyatta
Open vSwitch, Docker-simpleRouter

Figure 3.1 Technologies for virtualizing network functions with examples.

routers are mainly deployed in Points of Presence (PoP) of network carriers to save space and power, and reduce management overhead. Similarly, VLANs divide a physical switch into multiple logical switches by grouping ports on a switch. How a switch does grouping is implementation dependent, but a common solution is for the switch to tag each frame with a VLAN ID as it arrives on a port. When the frame is sent to another port, only the ports configured with the VLAN ID carried in the frame will output the packet. A VLAN can also span multiple interconnected switches using the IEEE standard 802.1Q. The limitation of VLAN is its low scalability, primarily due to a maximum of 4094 VLANs in a layer-2 network. To support a larger number of VLANs in a broadcast domain, VXLAN has been developed for large multi-tenant DC environments. In the optical domain, multiflow transponders can be used to create a number of subtransponders from the hardware resource pool [6]. These subtransponders can be used to carry different flows arriving from a single router interface by using flow identifiers.

Examples of soft partitioning include Virtual Routing and Forwarding (VRF) that allow multiple instances of routing and forwarding tables to co-exist within the same router. The various routing and forwarding tables may be maintained by a single process or by multiple processes (e.g. one process for each routing and forwarding table). Routing protocols should understand that certain routes may be placed only in certain VRFs. The routing protocols manage this by peering within a constrained topology, where a routing protocol instance in a VRF peers with other instances in the same VN. Another example of soft partitioning is FlowVisor that slices the flowspace of OpenFlow switches based on OpenFlow match fields, such as switch port, MAC addresses, and IP addresses. FlowVisor basically acts as a proxy between OpenFlow switches and controllers, and intercepts messages between them. By abstracting the OpenFlow control channel, FlowVisor provides mechanisms for bandwidth, switch CPU, and flowspace isolation.

3.2.1.2 Virtualized Network Functions

The main idea of VNFs is to decouple the physical network equipment from the functions that run on them. A VNF is an implementation of an NF that is deployed on virtual resources, such as a virtual machine (VM) or container [13]. A single VNF may be composed of multiple internal components, and hence it could be deployed over multiple VMs/containers, in which case each VM/container hosts a single component of the VNF.

For instance, a virtual router (vRouter) is a software function that implements the functionality of a layer 3 IP routing in software. The underlying physical resources are shared with other co-hosted VMs. In a well-implemented vRouter, users can see and change only the configuration and statistics for "their" router. Examples of vRouter include Alpine Linux, Mikrotik RouterOS, Brocade vRouter, Untangle, and Vyatta. Similarly, a virtual switch (vSwitch) is a software emulation of a physical switch that performs functions, such as traffic switching, multiplexing, and scheduling. It detects which VMs are logically connected to each of its virtual ports and uses this information to forward traffic to the correct VMs. Examples of vSwitch include Open vSwitch, Cisco Nexus 1000v, and VMware virtual switch. Due to the diversity of value-added NFs, different kinds of VNFs may exist based on different network layers. Even for each kind of NF, there may be multiple implementations with different features by various vendors. For example, the virtual NAT implemented by VMware provides a way for VMs to communicate with the host, while the one implemented by NFWare is extended to the carrier-grade level. For a comprehensive list of VNF products, the reader is referred to [14].

There are pros and cons of deploying a VNF on top of a VM or container. In case of VMs, the entire operational function of a VM is completely isolated from that of the host and other guest VMs. Hence, VM-based virtualization enforces a stronger isolation among VMs and the physical machine, and is regarded as a more secure and reliable solution. However, VM-based virtualization suffers from scalability and performance issues, due to the overhead of emulating a full computer machine within a VM. In contrast, containers do not need hardware indirection and run more efficiently on top of host OS, whereas each VM runs as an independent OS. Hence, containers can be used to deploy VNFs in a more flexible and agile manner but with a reduced level of isolation and security. Recently, unikernels have emerged as lightweight alternatives that take the best of both VM- and container-based virtualization. Unikernels usually package the VNFs with only the required libraries, unlike VMs that provide an entire guest OS.

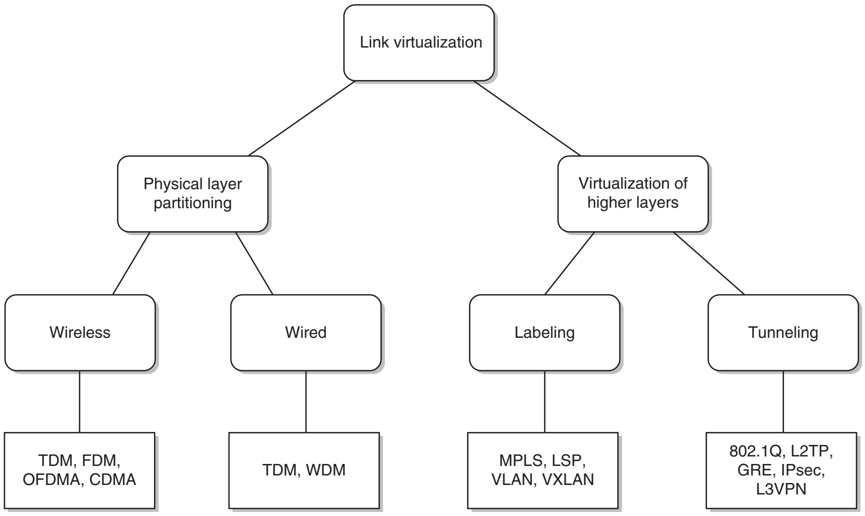

Figure 3.2 Link virtualization technologies with examples.

3.2.2 Link Virtualization

Link virtualization technologies enable creation of virtual links that can connect physical or virtual NFs. A virtual link can consist of a single physical link or can encompass a sequence of physical links forming a path. In this subsection, we discuss two popular technologies of virtualizing network links as follows (a summary is depicted in Figure 3.2).

3.2.2.1 Physical Layer Partitioning

Using various multiplexing technologies, a wired (e.g. fiber, copper cable) or wireless (e.g. wireless spectrum) physical medium can be split into distinct channels or time slots. A set of channels or time slots are then assigned to a virtual link with a specific bit rate such that the sender and receiver of the virtual link get the illusion that they own the physical medium. The type of multiplexing technique depends on the physical medium properties, the associated constraints and impairments. For example, a wireless link can be partitioned using time division multiplexing (TDM), frequency division multiplexing (FDM), or code division multiple access (CDMA). A combination of different multiplexing techniques can also be applied to achieve higher bandwidth, such as for broadband wireless networks. For example, orthogonal frequency-division multiple access (OFDMA) can be described as a combination of FDM and TDM multiple access, where the resources are partitioned in both time and frequency domains, and slots are assigned along the OFDM symbol index, as well as OFDM subcarrier index.

In fiber-optic communications, wavelength-division multiplexing (WDM) is a technology that multiplexes a number of optical carrier signals onto a lightpath (i.e. a set of concatenated optical fiber links) by using different wavelengths (i.e. colors) of laser light. This is similar to FDM, since wavelength and frequency communicate the same information. Physical layer multiplexing provides hard partitioning and better isolation among virtual links, since physical medium resources are assigned in a dedicated manner to virtual links.

3.2.2.2 Virtualization at Higher Layers

At higher layers (e.g. link, network, or application layers), link resource partitioning is achieved by allocating a specific bandwidth (i.e. transmission bit rate, link capacity) to a virtual link. Such partitioning can be enforced by rate-limiting or allocating an appropriate amount of link queues and link buffers. Since virtualization at higher layers is achieved through soft-partitioning of link resources, isolation between virtual links is especially critical. To ensure isolation among virtual links, two popular methods include: (i) labeling and (ii) tunneling.

Labeling involves specifying certain fields (e.g. tags, IDs, etc.) in the packet header that serve for identification and isolation of virtual links. For instance, VLANs apply tags to network packets and handle these tags in switches – creating the appearance and functionality of network traffic that is physically on a single network but acts as if it is split between separate VNs. VLANs can be used to distinguish data from different VLANs and to help form data paths for the broadcasting domain. Similarly, Multiprotocol Label Switching (MPLS) and label switched path (LSP) technologies can be used to specify the path that data packets take. In MPLS, labels identify virtual links (paths) between nonadjacent NFs. This requires MPLS capable routers (e.g. label-switched routers) to forward packets to outgoing interface based only on label value, unlike using IP addresses in traditional routers.

Tunneling is a popular method for link virtualization that has been adopted by many different technologies, such as VPN and VLAN. It ensures isolation of traffic from multiple VNs transported over a shared network. It also provides direct connection between network devices that are not physically adjacent. Tunneling is performed by using encapsulation and occasionally encryption techniques. A number of different tunneling technologies exist, including IEEE 802.1Q, Layer 2 Tunneling Protocol (L2TP), Generic Routing Encapsulation (GRE), Internet Protocol security (IPsec), and layer 3 virtual private network (L3VPN).

3.2.3 Network Virtualization

As discussed in the previous two subsections 3.2.2.1 and 3.2.2.2, both NFs and links can be independently virtualized while being oblivious to each other. It is also possible to virtualize only NFs and use non-virtualized links to connect VNFs and vice versa. In contrast, NV seeks to create slices of a network, i.e. VNs at the particular networking layer. For instance, a VN in the IP layer comprises of vRouters/vSwitch and overlay IP links connecting them, whereas a VN in the optical layer connects multiflow transponders through optical lightpaths. It should be noted that a given VN should have its own resources, including its own view of the network topology, its own portions of link bandwidths, dedicated CPU resources in NFs, and its own slices of CPU, forwarding/routing tables in switches and routers. Such a holistic NV can be achieved through network hypervisors that abstract the physical network (e.g. communication links, network elements, and control functions) into logically isolated VNs [11]. A number of network hypervisors, such as OpenVirteX, FlowVisor, OpenSlice, MobileVisor, RadioVisor, and Hyper-Flex, have been developed for different network technologies. The reader is referred to [11] for a more comprehensive survey of NV hypervisors.

3.2.4 Network Slicing

Network slicing extends the concept of NV in the context of 5G mobile networks from two perspectives. First, a 5G network slice is an end-to-end (E2E) VN that spans multiple technological and administrative network segments (e.g. wireless radio, access/core transport networks, Multi-access Edge Computing [MEC] and central DCs), whereas a traditional VN concerns only one particular network technology, such as wired transport or wireless network. Examples of network slices are shown in Figure 3.3 where the dark gray network slice goes all the way to the central DC, and dotted light gray network slice terminates at the central office of a mobile network. E2E perspective of network slices offer more opportunities to optimize the deployment of network slices, and meet fine-grained QoS requirements. Second, network slicing allows to virtualize RAN and core NFs, and include

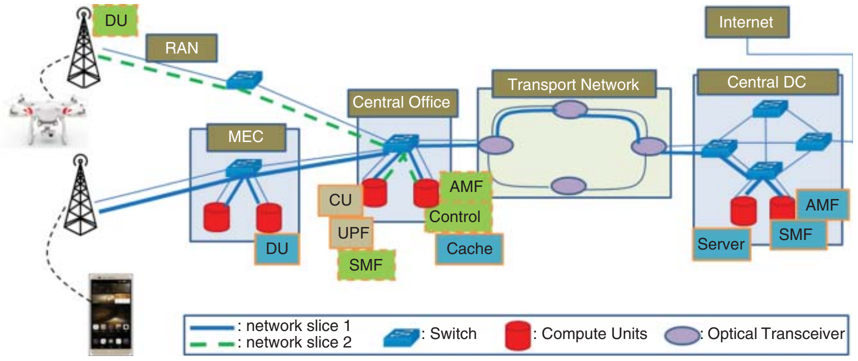

Figure 3.3 Examples of network slices.

them within a network slice that are typically not considered by conventional VNs. Virtualizing RAN and core NFs enable a more flexible way of creating, operating, managing, and deleting network slices on-demand. It also allows to deploy these VNFs with the appropriate capacity at the right place, to meet stringent requirements (e.g. E2E latency) imposed by 5G services.

Let us now discuss more about RAN and core NFs. The most common RAN functions responsible for baseband processing are: Service Data Adaptation Protocol (SDAP), Radio Resource Control (RRC), Packet Data Convergence Protocol (PDCP), Radio Link Control (RLC), Medium Access Control (MAC), and Physical (PHY) layer functions. In traditional mobile networks, Baseband Units (BBUs), co-located with antennas, are responsible for performing RAN NFs. However, in 5G RAN architecture, these NFs are envisioned to be virtualized and placed on commodity servers deployed either at antenna sites or MECs. Due to the strict timing requirements of some NFs, the RAN NFs are grouped in two entities: Central Unit (CU) and Distributed Unit (DU) [20]. DU hosts time-critical functions, such as MAC, RLC, and PHY, and serves a number of mobile users within the DU's coverage. On the other hand, CU may host time-tolerant functions, such as SDAP, PDCP, and RRC, and can serve multiple DUs. Both DU and CU can also be considered as aggregated VNFs and deployed on VMs/containers on servers located at antenna sites or MECs.

Similarly, a new core network architecture for 5G mobile networks, namely the Next Generation (NG) core, that separates the current Evolved Packet Core (EPC) functions into more fine-grained NFs has been proposed [20]. The most prominent NFs in NG core are as follows: Access and Mobility Management Function (AMF), Session Management Function (SMF), Policy Control Function (PCF), User Plane Function (UPF), and Unified Data Management (UDM). These core NFs can also be considered as VNFs and easily deployed in a virtualized environment. The benefit of this service oriented RAN and core architecture is that it allows for sharing of fine-grained VNFs among network slices without compromising the performance and QoS requirements. For instance in Figure 3.3, dark gray and dotted light gray network slices share CU and UPF NFs while using completely dedicated RAN and core NFs and their application functions (e.g. cache, control, or server). Similarly, the control plane functions, such as RLC, MAC, AMF, and PCF, can be shared between slices while using dedicated UPFs, including PDCP and UPF. Finally, the network slices that require the highest level of security (e.g. public safety or first responder's slice) may use dedicated VNFs not shared with others.

3.2.5 Management and Orchestration

SDN has the potential to simplify network configuration and reduce management complexity. In contrast to today's networks, where control and forwarding functions are tightly coupled and embedded within each network device (i.e. switches and routers), SDN accumulates the control functionality in a logically centralized and programmable control plane, which is decoupled from the forwarding plane. The control plane is implemented in software (i.e. SDN controller) on one or more dedicated computer servers, has a global network view, and provides a unified interface to configure and control the network. On the other hand, packet forwarding remains the responsibility of the switches/routers and is implemented on commodity hardware.

Management and Orchestration (MANO) is quintessential to unlock the full potential of NV, which includes seamless operation and efficient delivery of services. OpenStack is an open source cloud computing platform that controls large pools of virtual resources to build and manage private/public clouds. However, with the advent of NFV, OpenStack has become a crucial component in NFV MANO, as a Virtualized Infrastructure Manager (VIM). It is responsible for dynamic management of network function virtualization infrastructure (NFVI) hardware resources (i.e. compute, storage, and networking) and software resources (i.e. hypervisors), offering high availability and scalability. OpenStack also facilitates additional features in NFVI, including service function chaining and network slicing. Open Platform for Network Function Virtualization (OPNFV), a carrier-grade, open source platform also leverages OpenStack as its VIM solution [21].

Open Network Automation Platform (ONAP) and Open Source MANO (OSM) are two prominent NFV MANO initiatives. ONAP, a open source project hosted by the Linux Foundation, offers real-time, policy-driven orchestration of both physical and virtualized NFs, to facilitate efficient and automated delivery of on-demand services and support their lifecycle management. All ONAP components are offered as Docker containers, allowing for custom integration in different operator environments. It also allows for integration with multiple VIMs, VNF managers, and SDN controllers. ONAP primarily consists of two components: (i) design-time and (ii) run-time, each having subcomponents.

ONAP's design-time component offers a service design and creation (SDC) environment, that supports OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA), for describing resources and services (i.e. assets), along with their associated policies and processes. Its run-time component executes the policies prepared in the design-time, which pertain to monitoring, data collection, analytics, service orchestration, etc. ONAP leverages the Closed Loop Automation Management Platform (CLAMP), to enable lifecycle management of VNFs and automate E2E deployment processes. In contrast, OSM is an European Telecommunications Standards Institute (ETSI) initiative to offer cost-effective and automated delivery of services. Both ONAP and OSM conform to the ETSI NFV Reference architecture. A comparative evaluation of ONAP and OSM, with respect to features and performance gaps, is provided in [22]. Authors in [23, 24] propose an architecture for network slice management on top of ONAP, while [25] enhances OSM (along with OpenStack and OpenDaylight SDN controller) to enable service deployment across a multi-domain infrastructure.

3.3 State-of-the-Art

3.3.1 Network Virtualization

The embedding of VNs into substrate networks is a critical aspect of NV. The virtual network embedding (VNE) is a resource allocation problem that involves embedding virtual nodes and links to substrate nodes and links, respectively. For successful network embedding, it is paramount that resources are allocated efficiently. VNE is a well-studied problem that has been proved to be NP-hard [26, 27]. As a result, several linear programming algorithms, mixed integer programming algorithms, as well as heuristic algorithms have been proposed in the research literature. Most of the proposed heuristic algorithms solve the problem in two stages: (i) node embedding first and (ii) link embedding next. In the first stage, substrate nodes are ranked based on a specific metric (e.g. availability) and a greedy node mapping strategy is applied where mapping is decided by rank results. In the second stage, the virtual links are usually mapped to the shortest path that has enough bandwidth resources between nodes. On the other hand, linear programming and mixed integer programming algorithms are used to solve the VNE problem in a single stage, by simultaneously mapping nodes and links.

The majority of VNE solutions perform static mappings and resource allocations i.e. they do not consider the remapping of embedded VNs by migrating virtual nodes and/or links or adjusting the resource allocated to the VN, as new requests are received, or the network load, traffic pattern changes. Indeed, this is counterintuitive, considering the dynamic nature of Internet traffic. The proven inefficiency of static resource allocation motivated the emergence of dynamic solutions. ML, in particular reinforcement learning (RL), have been proven particularly efficient for solving the dynamic resource allocation problem, considering the higher complexity of the problem compared to static VNE. Table 3.1 provides a summary of the state-of-the-art that addresses VNE and resource allocation.

Mijumbi et al. [28] address the dynamic resource allocation problem using an RL-based approach. They model the substrate network as a decentralized system of Q-learning agents, associated to substrate nodes and links. The agents use Q-learning to learn an optimal policy to dynamically allocate network resources to virtual nodes and links. The reward function encourages high virtual resource utilization, while penalizing packet drops and high delays. The agents ensure that

Table 3.1 Summary of the state-of-the-art for virtual network embedding.

References	Problem/objective	Features	ML technique
[28]	Dynamic resource allocation to achieve high resource utilization and QoS	Virtual resource substrate resource	RL with Q-learning
[29]	Dynamic resource allocation to achieve high resource utilization and QoS	Virtual resource substrate resource	RL with ANN
[30, 31]	Node mapping to achieve high revenue-to-cost ratio	CPU bandwidth topological features	RL with ANN
[32]	Node mapping to achieve high revenue-to-cost ratio	CPU bandwidth degree	RL with RNN
[33]	VNE admission control	CPU bandwidth topological features	RNN
[34]	Substrate subgraph extraction to speed up VNE process	CPU bandwidth topological features	Hopfield network
[35]	Node mapping to achieve high acceptance ratio, high revenue-to-cost ratio, and load balancing	CPU bandwidth embedding status	Deep RL with GCN

while the VNs have the resources they need, at any given time only the required resources are reserved for this purpose. Simulations show that the RL-based dynamic resource allocation significantly improves the VN acceptance ratio, and the maximum number of accepted VN requests at any time, in comparison to the static approach. The approach also ensures that VN's QoS requirements, such as packet drop rate and virtual link delay, are not affected.

In a subsequent work [29], Mijumbi et al. leverage artificial neural networks (ANNs) and propose an adaptive resource allocation mechanism, which unlike the Q-learning-based solution in [28], does not restrict the state-action space. Similar to [28], resource allocation decisions are made in a decentralized fashion by RL agents associated to each substrate node and link. Each agent relies on an ANN whose input is the status of the substrate node (respectively link) and embedded virtual nodes (respectively links), and that outputs an allocation action. An error function that evaluates the desirability of the ANN output is used for training purposes. The objective of the error function is to encourage high virtual resource utilization, while penalizing packet drops and high delays. Simulations show that the ANN-based RL solution outperforms the *Q*-learning-based solution, which is attributed to a state-action space expressed at a finer granularity.

In [30, 31], Yao et al. build on the intuition that network requests follow an invariable distribution, such that if an embedding algorithm works well for historical VN requests, it is likely to have the same performance for incoming VN requests. They propose in [30] a two-phased VNE algorithm i.e. a policy gradient RL-based node-mapping phase, followed by a breadth-first search for the shortest paths between the selected host nodes in the link-mapping phase. The node-mapping agent is implemented as an ANN. It is trained with historical network data and tuned using policy gradient based on the average revenue-to-cost ratio metric. The agent's goal is to observe the current status of the substrate network and output node mapping decisions. The status of the substrate network is represented by a matrix that combines topological features and resource usage extracted from every substrate node. In [31], this matrix is further reduced using a spectrum analysis method. The reduced matrix is combined with a reduced form of the substrate network adjacency matrix. Perturbation is applied to the resulting matrix every time an embedding occurs, in lieu of systematic updates for reduced complexity. Simulations show that the model devised in [31] outperforms the original model from [30].

More recently, Yao et al. [32] explore replacing the ANN node-mapping agent with a Recurrent Neural Network (RNN), after formulating VNE as a time-series problem. The intuition is that node embedding is a continuous decision process. The RNN agent, implemented as a seq2seq model, is trained with historical network data and fine tuned using the policy gradient algorithm based on the long-term average revenue-to-cost ratio metric. Simulation results show an improvement compared to the original model from [30] in terms of request acceptance ratio, long-term revenue and long-term revenue-to-cost ratio.

In [33], Blenk et al. study the online VNE satisfiability problem. They propose an RNN-based classifier that, for a given VN request, outputs whether the embedding is possible or not. The model is meant to run prior to the VNE algorithm per se, as an admission control procedure. The goal is to save time and resources that might be wasted trying to satisfy an embedding request that cannot be satisfied, at least not in an acceptable time, in the current state of the substrate network. The authors additionally devise a novel, relatively low-complexity representation of the substrate network, as well as VN requests that combine topological features and resource usage. Simulations show that their classifier is highly accurate and significantly reduces the overall computational time for the online VNE problem, without severely impacting the performance of embedding.

In their continued effort to speed up and improve rigorous online VNE algorithms, Blenk et al. [34], leverage Hopfield networks to devise a VNE preprocessing mechanism that performs search space reduction and candidate subgraph extraction. More precisely, the designed Hopfield network computes a probability for each substrate node to be part of the candidate subgraph for a given embedding request. A rigorous VNE algorithm is then used to find the final embedding solution within the extracted subgraph. Simulations show that the proposed preprocessing step improves the runtime and/or performance of most of the tested online VNE algorithms, depending on the parameters of the Hopfield network, which have to be determined beforehand.

Yan et al. [35] build on recent advancements in deep learning and propose a deep RL solution to the node mapping problem, to reduce the overall runtime of the VNE algorithm. The authors focus on the static allocation of substrate resources. They use Graph Convolutional Networks (GCN), for the learning agent to extract spatial features in the substrate network and find the optimal node mapping. The learning agent is trained using a parallel policy gradient approach, which is shown to converge faster and perform better than sequential training. In addition to rewarding higher acceptance ratio and revenue-to-cost ratio, the used reward signal also encourages policy exploration and is shown to lead to higher performance than more traditional reward functions. The proposed deep RL solution is shown to outperform state-of-the-art non-ML embedding algorithms.

3.3.2 Network Functions Virtualization

3.3.2.1 Placement

Placement of SFCs can have varying objectives, such as minimizing the cost of placement, cost of operation (e.g. licensing fee, energy consumption), service-level agreement (SLA) and QoS requirements. This problem is known to be NP-hard, making it difficult or even prohibitive to solve it optimally for large problem instances. Furthermore, heuristics tend to be inefficient in the face of high number of constraints and changes in network dynamics [36, 37]. Recently, RL has been explored to facilitate SFC placement in virtualized environments. Traditional RL maintains a Q-table to store policies (i.e. Q-values), and the RL agent uses feedback from the environment to learn the best sequence of actions or policy to optimize a cumulative reward. However, it does not scale for large state-action space [38]. In contrast, deep RL leverages Neural Networks (NNs) to learn the Q-function that map states, actions to Q-values. Deep RL can be classified into value-based, such as deep Q-learning network (DQN), and policy-based approaches. Table 3.2 provides a summary of the state-of-the-art that addresses NFV placement.

In [39], Pei et al. translate QoS requirements as a penalty when failing to serve a service-function chain request (SFCR) in VNF placement. They employ double deep Q-learning network (DDQN) that includes two NNs, one for selecting state, action and the other for evaluating the Q-value. Once the DDQN has been trained,

Table 3.2 Summary of the state-of-the-art for ML-based placement in NFV.

References	Problem/objective	Features	ML technique
[39]	Minimize operational cost and penalty for rejecting SFCR	CPU, memory, bandwidth	Deep RL with DDQN
[40]	Minimize cost of provisioning VNFs on multi-core servers for SFCRs	CPU	RL with Q-learning and ϵ-greedy policy
[36]	Maximize the number of SFCs based on QoS requirements	CPU, memory, storage, bandwidth	Deep RL with DDPG and MCN
[37]	Minimize infrastructure power consumption	CPU, storage, bandwidth, propagation delay	NCO with stacked LSTM and policy gradient
[38]	Minimize operational cost and maximize QoS w.r.t. total throughput of accepted SFCR	CPU, memory, bandwidth, latency	Deep RL with policy gradient
[41]	Minimize discrepancy in predicted and actual total response time	Transmission, propagation, processing times, CPU, storage	RL with Q-learning and ϵ-greedy policy

it can be used for VNF placement. Each action has an associated reward that reflects the influence of the action on the network. After deployment, the DDQN evaluates the performance of the actions and selects the highest reward action according to a threshold-based policy, to trigger horizontal scaling. After VNF placement, the authors use SFC-MAP [42] to construct the routing paths for the ordering required in the SFCRs.

In contrast, to avoid expensive bandwidth consumption, Zheng et al. [40] jointly optimize the cost of provisioning VNFs on multi-core servers (i.e. VNF assignment to CPU core). However, there is still unpredictability in VNF deployment, such as the random arrival of SFCRs, resources consumed and cost of provisioning. The authors leverage Q-learning to alleviate the need to know the state transitions a priori. They employ value iteration to select a uniform, random action, implement it, and evaluate the reward. In this way, their approach updates the Q-table to identify the state transitions and be resilient in the face of changing rate of SFCRs. The authors leverage an ϵ-greedy algorithm that strikes a balance between exploration and exploitation, and control the influence of historical experience. On the other hand, Quang et al. [36] employ deep Q-learning (DQL) to maximize the number of SFCs on a substrate network, while abiding by infrastructure constraints. They leverage deep deterministic policy gradient (DDPG), where deep NNs (DNNs) i.e. actor and critic, separately learn the policy and Q-values, respectively. The authors improve DDPG by using multiple critic network (MCN) for an action, where the actor NN is updated with the gradient of the best critic in the MCN, thus improving convergence time.

The Neural Combinatorial Optimization (NCO) paradigm is extended by Solozabal et al. [37], to optimize VNF placement. Their NCO leverages an NN to model the relationship between problem instances (i.e. states) and corresponding solutions (i.e. actions), where the model weights are learnt iteratively via RL, specifically policy gradient method. Once the RL agent converges, given a problem instance, it returns a solution. This allows to infer a placement policy for a given SFCR that minimizes the overall power consumption of the infrastructure (i.e. the cost function or reward), given constraints, such as availability of virtual resource and service latency thresholds. The constraints are incorporated into the cost function using Lagrange relaxation, which indicates the degree of constraint dissatisfaction. For NN, the authors employ stacked Long Short-Term Memory (LSTM), which allows to accommodate for SFCs of varying sizes. The authors show that the proposed agent when used in conjunction, improves the performance of the greedy First-Fit heuristic.

In [38], Xiao et al. jointly address the following SFC deployment challenges: (i) capturing the dynamic nature of service request and network state, (ii) handling the different network service request traffic characteristics (e.g. flow rate) and QoS requirements (e.g. bandwidth and latency), and (iii) satisfying both provider and customer objectives i.e. minimize operation cost and maximize QoS, respectively. For the first challenge, the authors leverage Markov Decision Process (MDP) to model the dynamic network state transitions, where a state is represented as the current network resource utilization (i.e. CPU, memory, and bandwidth) and impact of current SFCs, while the action corresponds to the SFC deployment corresponding to an arriving service request. For the second challenge, the authors employ policy gradient based deep RL to automatically deploy SFCs. After RL convergence, it provides SFC deployment solution to each arriving request, abiding by resource constraints. They address the third challenge by jointly maximizing the weighted total throughput of accepted service requests (i.e. income) and minimizing the weighted total cost of occupied servers (i.e. expenditure), as the MDP reward function (i.e. income minus expenditure). Via trace-driven simulation, the authors show their approach to outperform greedy and Bayesian learning-based approaches, providing higher throughput and lower operational cost on average.

Bunyakitanon et al. [41] define E2E service level metrics (e.g. VNF processing time, network latency, etc.) in support of VNF placement. They account for heterogeneous nodes with varying capabilities and availability. The authors purport that their Q-learning based model generalizes well across heterogeneous nodes and dynamically changing network conditions. They predict the service level metrics and take actions that maximize the reward for correct predictions. The Q-values are updated using a weighted average of new and historical Q-values. The reward incorporates an acceptable margin of error, with the highest reward for predicting a value that equals the actual value. They employ an ϵ-greedy policy to strike a balance between exploration and exploitation, starting with an equal probability to explore or exploit. Then, they generate a random number, and compare it to the ϵ-greedy value to steer toward exploration or exploitation. The authors show that their model has the best performance with approximately 94% in exploration and 6% in exploitation.

3.3.2.2 Scaling

VNF resource scaling assumes an initial deployment of SFCs, with the primary objective of accommodating for the changes is service demand. Static threshold-based scaling is relatively simple to implement, where predefined thresholds are used per performance metric, such as CPU utilization, bandwidth utilization, etc. For example, Ceilometer, in OpenStack Heat, can be used to create alarms based on CPU utilization thresholds to spin up or terminate virtual network function instances (VNFIs) [43]. However, it is not only nontrivial to choose these thresholds, they may also require frequent updates to accommodate for the varying service requirements. Table 3.3 provides a summary of the state-of-the-art that addresses NFV scaling.

Static threshold-based scaling is reactive and unable to cope with sudden changes in service demand, leading to resource wastage and SLA violations. Moreover, over provisioning can lead to low resource utilization and high operational costs, while under provisioning can result in service disruption and even outage. Tang et al. [44] propose an alternative to static threshold-based scaling mechanisms, which is SLA-aware and resource efficient. They model VNF scaling as an MDP and leverage Q-learning to decide on the scaling policy. In the evaluation on daily busy-and-idle and bursty traffic scenarios, their approach outperforms static threshold-based and voting policy-based (e.g., majority of the performance metrics have to agree to a scaling action, based on their respective thresholds) approaches, while striking a trade-off between SLA guarantee for network services and VNF resource consumption.

Proactive scaling leverages service demand and/or threshold predictions to dynamically allocate resources to SFCs. ML is an ideal technique to perform predictions based on historical data, while ML features play a pivotal role in its performance. Cao et al. [45] use novel ML features for scaling, which include VNF and infrastructure level metrics. They train an NN on labeled data, to capture the complex relationships between resource allocation, VNF performance, and

Table 3.3 Summary of the state-of-the-art for ML-based scaling in NFV.

References	Problem/objective	Features	ML technique
[44]	Trade-off between SLA and VNF resource consumption	CPU, memory, storage, bandwidth, network users and requests	RL with *Q*-learning
[45]	Learning resource allocation and VNF performance relationship	VNF internal statistics (e.g. request queue size) and resource utilization	NN, decision table, random forest, logistic regression, naïve bayes
[46]	Meet service demands	Performance measurements (e.g. max sustainable traffic load) and resource requirements (e.g., CPU, memory)	Support vector regression, decision tree, multi-layer perceptron, linear regression, ensemble
[47]	Predict VNFC resource reliability	QoS requirements	Bayesian learning
[43]	Predict VNFC resource reliability	CPU, memory, link delay	GNN with FNNs
[48]	Predict VNFIs, and minimize QoS violations and operational cost	Time of day, measured traffic load at different time units, and changes in traffic	Multi-layer percep., bayesian network, reduced error pruning tree, random and C4.5 decision trees, random forest, decision table
[49]	Minimize average oper. cost, SLA violation and VNF latency w.r.t. resizing, deployment, off-loading	CPU, memory, QoS	Deep RL with twin delayed DDPG and DNN

service demand. However, labeling is not only cumbersome, tedious, and error prone but it also requires NFV domain expert knowledge. The authors prioritize resource allocation for VNFs based on urgency and attempt to distribute load across all instances of the VNF, using traffic forwarding rules. However, if existing instances of a VNF cannot meet the service demand, new instances must be spawned using VNF placement algorithms. While Cao et al. [45] show the benefit of composite features (i.e. VNF and infrastructure level), Schneider et al. [46] promote the use of ML for creating performance profiles that precisely capture the complex relationships between VNF performance and resource requirement. On the other hand, Shi et al. [47] leverage MDP to scale virtual network function components (VNFCs). To improve MDP performance, the authors employ

Bayesian learning and use historical resource usage of VNFCs to predict future resource reliability. These predictions are leveraged in an MDP to dynamically allocate resources to VNFCs, and facilitate system operation without disruption. Their approach outperforms greedy methods in overall cost.

Mijumbi et al. [43] draw logical relationships among VNFCs in a SFC, to forecast future resource requirements. The novelty lies in identifying relationships among VNFCs that may or may not be ordered within a VNF. The authors leverage graph NN (GNN) to model each VNFC in the SFC as two parametric functions, each modeled as a feedforward NN (FNN). These pairs of FNN are responsible for learning the resource requirements of the VNFC, using historical resource utilization information from the VNFC and its neighboring VNFCs (i.e. using the first FNN), followed by prediction of future resource requirements of the VNFC (i.e. using the second FNN). The authors employ backpropagation-through-time to update the NN weights and improve prediction performance. Similar to [45], they also leverage VNF (e.g. CPU utilization, memory, processing delay) and infrastructure level (e.g. link, capacity, latency) features. Their model yields the lowest mean absolute percentage error, when the prediction window size is within the training window size. Otherwise, the prediction accuracy suffers, requiring model retaining.

In [48], Rahman et al. use traffic measurements and scaling decisions across a time period to extract features and define classes for ML classifiers (e.g. random forest, decision table, multi-layer perceptron, etc.). The features represent measured service demand and its change from recent history, while classes represent the number of VNFIs. These features and classes are used to train ML classifiers and predict future scaling decisions. The authors leverage two classifiers, the first predicts scaling to avoid QoS violations, while the other predicts scaling to reduce operational cost. In the face of inaccurate scaling predictions and/or delays in VM startup time, ML classifiers trained to reduce QoS violations stay in a state of degraded QoS for shorter periods of time. Containerization has been shown to reduce startup times for VNFIs and significantly improve QoS.

Roig et al. [49] use unlabeled data to decide on vertical, horizontal scaling or offloading to a cloud, based on service requests, operational cost, QoS requirements and end-user perceived latency. The authors employ a parameterized action MDP, where a set of continuous parameters are associated with each action. The actions correspond to the user-server assignment, while the parameters identify the scaling of VNF server resources (i.e. compute and storage). This allows for selecting different servers for users requesting the same VNF service to increase sensitivity to end-user perceived latency and enable asynchronous manipulation of server resources. The authors leverage deep RL that parameterizes the policy, and employ actor and critic NNs to learn the policy using a twin delayed DDPG. A DNN is used to approximate the policy that optimizes the weighted average of latency, operational cost, and QoS. Since the weights can be adjusted and used to update the policy, it not only performs well under constant service demand but it also quickly adapts to variation in service requests and is resilient to changes in network dynamics.

3.3.3 Network Slicing

3.3.3.1 Admission Control

Admission control dictates whether a new incoming slice request should be granted or rejected based on available network resources, QoS requirements of the new request and its consequence on the existing services, and ensuring available resources for future requests. Evidently, accepting a new request generates revenue for the network provider. However, it may degrade the QoS of existing slices, due to scarcity of resources, consequentially violating SLA and incurring penalties, loss in revenue. Therefore, there is an inherent trade-off between accepting new requests and maintaining or meeting QoS. Admission control addresses this challenge and aims to maximize the number of accepted requests without violating SLA. Several research efforts, as outlined below, have addressed the slice admission control problem from different perspectives using ML. Table 3.4 provides a summary of the state-of-the-art for ML-based admission control approaches in network slicing.

Bega et al. [50] present a network slice admission control algorithm that maximizes the monetization of the infrastructure provider, while ensuring slice SLAs. The algorithm achieves the objective by autonomously learning the optimal admission control policy, even when slice behavior is unknown and data is unlabeled. The authors consider two types of slices: (i) inelastic, whose throughput should always be above the guaranteed rate, and (ii) elastic, whose throughput is allowed to fall below the guaranteed rate during some periods, as long as the average stays above the specified rate. Since the type of the slice, its arrival and

Table 3.4 Summary of the state-of-the-art for ML-based admission control approaches in network slicing.

References	Problem/objective	ML technique
[50]	Maximize monetization of infrastructure provider, while ensuring slice SLAs	Deep RL framework with two different NNs
[51]	Minimize loss of revenue and loss due to penalties in service degradation	Resource prediction and RL
[52]	Maximize resource utilization while respecting slice priorities	RL with Q-learning

departure are unknown in advance, it is impossible to establish the ground truth for the admission control problem. Therefore, the authors propose a deep RL approach that interacts with the environment and takes decision at a given state, while receiving feedback from past experiences. Their deep RL framework uses two different NNs, one to estimate the revenue for each state when accepting the slice request, and another to reject the request. The framework then selects the action with the highest expected revenue, and the reward for the action is fed back to RL. Through evaluation, the authors show that their proposed algorithm performs close to the optimal under a wide range of configurations, and outperforms naïve approaches and smart heuristics.

Raza et al. [51] address the network slice admission control problem by taking into account revenues of accepted slices, and penalties proportional to performance degradation, if an admitted slice cannot be scaled up later due to resource contention. The authors propose a supervised learning (SL)- and a RL-based algorithm for slice admission control. The SL-based solution leverages prediction for the incoming slice requirement, and future changes in requirement for the incoming slice and all other slices currently provisioned. This facilitates identification of possible degradation in performance upon admission, for the incoming slice or currently provisioned slices, which results in slice rejection. On the other hand, the RL-based algorithm learns the relationship between slice requirement and current resource allocation, along with the overall profit. This relationship guides slice admission policy, allowing to only accept slices that are likely to experience/create minimal to no degradation in performance. The objective of the admission policy is loss minimization, where the loss has two components: (i) loss of revenue due to rejecting slice requests, and (ii) the loss incurred due to penalties in service degradation, as described in [53].

An RL-based solution for cross-slice congestion control problem in 5G networks, which impacts the slice admission control process, is proposed by Han et al. [52]. Their solution identifies active slices with loose requirements i.e. their amount of allocated resources can be reduced based on resource availability, slice requirements, and the queue state. The identified slices' resources are then scaled down, to make room for a larger number of higher priority slices. To achieve this, the authors use Q-learning that can learn optimal resource reallocation strategy, by jointly maximizing resource utilization and respecting slice priorities. The evaluation results show that the proposed solution is able to increase the percentage of accepted slice requests, without negatively affecting the performance of high priority slices.

3.3.3.2 Resource Allocation

An E2E network may simultaneously require radio, network, computing, and storage resources from multiple network segments. An emerging challenge for

Table 3.5 Summary of the state-of-the-art for ML-based resource allocation approaches in network slicing.

References	Resource type	Problem/objective	ML technique
[54]	Virtual protocol stack functions, RRU association, sub-channel and power allocation	Maximize service utility in terms of the difference between revenue and expense	RL with ϵ-greedy Q-learning
[55]	VMs and bandwidth	Minimize processing delays for received requests and resource usage costs	RL with policy gradient methods
[56]	Service capacity requirement	Maximize revenues in short- and long-term resource reallocation	ANN-based deep learning prediction
[57]	Slice bandwidth allocation and scheduling of SFC flows	Maximize the weighted sum of spectrum efficiency and QoE	RL with Deep Q-Learning
[58]	Bandwidth or time-slots	Maximize SSR and spectrum efficiency	Dueling GAN-DDQN
[59]	Computing, storage, and radio resources	Maximize the long-term average reward	RL (Q-learning, DQL, deep double Q-learning, and deep dueling)

the network provider is how to concurrently manage multiple interconnected resources. Due to the dynamic demand of services, the frequency of slice requests, their occupation time, and requirements are not known a priori, while the resources are limited. Hence, dynamically allocating resources in real-time to maximize a specific objective is another challenge for the network provider. Table 3.5 provides a summary of the state-of-the-art for ML-based resource allocation approaches in network slicing.

Wang and Zhang [54] propose a two-stage network slice resource allocation framework based on RL (i.e. Q-learning). The first stage performs the mapping of virtual protocol stack functions of a network slice to physical server node. The second stage manages remote radio unit (RRU) association, sub-channel, and power allocation. Instead of applying one Q-learning model to solve the joint problem, the authors use two ϵ-greedy Q-learning models sequentially, to keep the model scalable. The optimization goal of the proposed model is to maximize the service utility (i.e. difference between revenue and expenditure) of the whole network, where the revenue comes from the service rate, and the expenditure comes from the virtual function deployment cost and E2E delay loss. Simulation results show that compared to the baseline schemes (e.g. minimum cost function deployment and radio resource allocation maximizing signal to noise ratio), the proposed algorithm can increase the utility of the whole system. However, there is an upper limit, due to the limited node resources, while simulation is performed only on a few tens of users in the system.

A deep RL approach is proposed by Koo et al. [55], which addresses the network slice resource allocation problem by considering unknown slice arrival characteristics, and heterogeneous SLA and resource requirements (e.g. VMs, bandwidth, and memory). The slice resource allocation pertains to allocating VMs and bandwidth for each slice, with the objective of minimizing processing delays for received requests and resource usage costs. The authors formulate the resource allocation problem as an MDP, where the constrained multi-resource optimization problem is formulated for each service upon arrival and a batch of services. For both types of request, RL models are trained offline to learn efficient resource allocation policies, which are used in real-time resource allocation. The policies are stochastic, and determine real valued resource allocations for each slice that has large and continuous action space. The authors use policy gradient methods as opposed to Q-learning, which cannot represent stochastic and continuous action spaces. Simulations using both simulated and real traces show that the model outperforms a baseline of equal-slicing strategy, which fairly divides the resources among each slice.

Bega et al. [56] present DeepCog, a data analytics tool for cognitive management of resources in 5G network slices. DeepCog forecasts the capacity needed to accommodate future traffic demands of individual network slices, while accounting for the operator's desired balance between resource over provisioning (i.e. allocating resources exceeding the demand) and SLA violations (i.e. allocating less resources than required). DeepCog uses an ANN-based deep learning prediction mechanism that consists of an encoder with three layers of three-dimensional convolutional NNs and a decoder implemented by multi-layer perceptrons. The encoder–decoder structure is shown to predict service capacity requirement with high accuracy, based on measurement data collected in an operational mobile network. The authors claim that the structure is general enough to be trained to solve the capacity forecast problem for different network slices with diverse demand patterns. The capacity forecast returned by DeepCog, can then be used by operators to take short- and long-term resource reallocation decisions and maximize revenues.

In [57], Li et al. address resource management for network slicing independently for radio resource slicing and priority-based core network slicing. In the radio part, resource management pertains to slice bandwidth allocation to maximize the weighted sum of spectrum efficiency and quality of experience (QoE). For the core network slicing, the goal is to schedule flows to SFCs that incur acceptable waiting times. For both of these problems, the authors leverage DQL to find the optimal resource allocation policies, which enhance effectiveness and agility of network slicing in a resource-constrained scenario. However, their approach does not consider the effects of random noise on the calculation of spectrum efficiency and QoE for radio resource slicing. To overcome this limitation, Hua et al. [58] combine distributional RL and Generative Adversarial Network (GAN), to propose GAN-powered deep distributional *Q* network (GAN-DDQN). Furthermore, the authors adopt a reward-clipping scheme and introduce a dueling structure to GAN-DDQN (i.e. Dueling GAN-DDQN), to separate the state-value distribution and the action advantage function from the action-value distribution. This circumvents the inherent training problem of GAN-DDQN. Simulation results show the effectiveness of GAN-DDQN and Dueling GAN-DDQN over the classical DQL algorithms.

Van Huynh et al. [59] propose a resource management model that allows the network provider to jointly allocate computing, storage, and radio resources to different slice requests in a real-time manner. To deal with the dynamics, uncertainty, and heterogeneity of slice requests, the authors adopt semi-MDP. Then, several RL algorithms, i.e. *Q*-learning, DQL, deep double *Q*-learning, and deep dueling, are employed to maximize the long-term average reward for the network provider. The key idea of the deep dueling algorithm is to use two streams of fully connected hidden layers to concurrently train the value and advantage functions, thus improving the training process. Simulation results show that the proposed model using deep dueling can yield up to 40% higher long-term average reward, and is a few thousand times faster compared to other network slicing approaches. The advantage of the proposed model is that it can accommodate adding more resources or removing some resources (i.e. scaling out or scaling in, respectively) by considering some new events in the system state space. However, the work of [59] overlooks the network resources that is needed for an E2E slice provisioning.

3.4 Conclusion and Future Direction

Virtualized networks and services bring inherent challenges for network operators, which calls for automated management that cannot be satisfied with the traditional, reactive human-in-the loop management approach. Furthermore, the requirement for higher QoS and ultra-low latency services necessitates intelligent management that should harness the sheer volume of data within a network, and take automated management decisions. Therefore, AI and ML can play a pivotal role to realize the automation of management for virtualized networks and services. In Section 3.3, we discuss the state-of-the-art in employing AI and ML techniques to address various challenges in managing virtualized networks and services, specifically in NV, NFV, and network slicing. In this section, we delineate open, prominent research challenges and opportunities for holistic and automated management of virtualized networks and services.

3.4.1 Intelligent Monitoring

Monitoring requires the identification of Key Performance Indicators (KPIs), such as perceived latency, alarms, and utilization of virtualized network components [60]. These play a crucial role in analytics to facilitate automated decision-making for managing virtualized networks and services. It is quintessential that the employed measurement techniques collect telemetry data with high accuracy, while minimizing overhead. However, measurement can add significant overhead (e.g. consumed network bandwidth, switch memory due to probing, and storage) when a large number of virtualized network components are monitored at regularly occurring intervals. This instigates the need for adaptive measurement schemes that can dynamically tune monitoring rate and decide what to monitor. ML techniques, such as regression, can facilitate adaptive monitoring by predicting telemetry data that would have otherwise been measured. Another challenge is to devise mechanisms for timely and high precision instrumentation to monitor KPIs for virtualized networks with demanding QoS requirements, especially for ultra-low latency services.

3.4.2 Seamless Operation and Maintenance

ML-based predictive maintenance can enable seamless operation of virtualized networks [19]. It involves inferring future events based on measured KPIs, identifying causes of performance degradation, and proactively taking preventive measures. An example of inference is to determine if a performance degradation (e.g. increased packet loss, prolonged downtime) would lead to future QoS violations. It is also crucial to infer causes (e.g. misconfiguration, failure) of performance degradation in correlation with potential alarms. However, realizing this from the enormous volume of telemetry data and stochastic nature of network events is challenging. Data-driven approaches, including ML, can be explored to address these problems. Once the cause for performance degradation is identified, mitigation workflows are needed to minimize the impact on KPIs. Deducing these workflows and optimally scheduling their execution with minimal interruption to the existing traffic is nontrivial. However, RL seems well suited to the problem and should be investigated to find optimal mitigation workflows.

3.4.3 Dynamic Slice Orchestration

In 5G mobile networks, an E2E VN slice spans multiple network segments, each of which can have different technological and physical constraints. For instance, the access network may have limited bandwidth and scalability to minimize cost and energy, while the core network may not have these issues of capacity or scalability. However, the core network may have higher latency and energy footprint due to long geographical distances and more complex network devices. Similar trade-offs exist between edge and central DCs, with respect to processing capacity, latency, and energy consumption. Therefore, it will be impractical to provision a network slice for its peak traffic demand. Hence, dynamic slice provisioning algorithms must be investigated, where resource orchestration decisions are facilitated by ML models for slice traffic volume prediction with temporal, spatial considerations and QoS requirements. Such dynamic slice provisioning will be enabled by NFV that allows for spawning on-demand virtualized NFs, and SDN controllers that can route traffic to newly spawned NFs.

3.4.4 Automated Failure Management

Even with predictive maintenance, some failures, such as fiber cuts and device burns are inevitable. Ability of a network provider to quickly repair a failure is crucial to keep the network operational. Failure management involves three steps: failure detection, localization, and identification. The goal of failure detection is to trigger an alert after the failure has occurred. Once detected, the failed element (e.g. the node or link responsible for the failure) must be localized in the network to narrow down the root cause of failure. Even after localization, it might still be complex to understand the exact cause of the failure. For example, inside a network node, the degradation can be due to misconfiguration or malfunction. To speed up the failure repair process, all three steps of failure repair should be automated. An interesting avenue of research is to develop ML models and algorithms for automated failure detection, localization, and identification based on the data generated in production networks. These models will decrease the mean time to repair after failure events, thus improving the availability of a network slice or a virtualized network/service.

3.4.5 Adaptation and Consolidation of Resources

The traffic demand and/or QoS requirement of a virtualized network or a network slice may evolve over time, due to change in number of users and communication patterns [61]. Hence, the initial resource allocation need to be adapted to accommodate for such changes, while causing minimal to no disruption to existing traffic. This calls for ML models to predict change in requirements in a timely and accurate manner, and facilitate dynamic adaptation of resource allocation. Furthermore, over time, arrival and departure of virtualized networks or network slices can lead to fragmentation and skewed utilization of links and processing servers. These, in turn, can impact the acceptance of future requests and result in unnecessary energy consumption. One way to mitigate this is by re-optimizing bandwidth allocation and periodically consolidating VMs or containers. The solution should also output the sequence of operations (e.g. VM migration, virtual link migration, and bandwidth reallocation) that lead to a load-balanced state. RL is an ideal technique to generate the sequence of operations needed to reach the optimized state.

3.4.6 Sensitivity to Heterogeneous Hardware

In NFV deployment or in network slices, VNFIs that reside on VMs or containers are scaled to meet the service demands. However, the performance of VNFs is sensitive to the underlying hardware [45, 46]. For example, traffic processing capabilities of virtual CPUs on Intel Xeon processor differ from AMD Opteron processor [45]. Similarly, boot up time for VMs differ across VIMs, such as OpenStack, Eucalyptus, and OpenNebula [43]. Nevertheless, most research assumes homogeneous hardware, being oblivious to its impact on VNF performance. This is an oversimplification, which can lead to inferior ML models and inaccurate scaling decisions in practice. Therefore, it is quintessential to develop performance profiles [46], which incorporate the sensitivity of VNF performance on different hardware. In case of horizontal scaling, these profiles can be leveraged to accurately gauge the impact on performance for new VNFIs on different physical servers. Indeed, incorporating these profiles will increase the dimensionality of the scaling problem. A naïve option is to incorporate hardware-sensitivity as a cost. However, building VNF performance profiles for different hardware is cumbersome. It remains to be evaluated how these hardware-specific performance profiles will impact the accuracy of ML models and VNF scaling decisions.

3.4.7 Securing Machine Learning

Evidently, there has been a surge in the application of ML for managing virtualized networks, ranging from placement and scaling of VNFs to admission control in network slices. However, numerous research assumes ML itself to be invincible. This is an unrealistic assumption, as adversaries can poison the training data, or compromise the RL agent by manipulating system states and policies, leading to inferior actions [62]. For example, impeding actual resource consumption of substrate network can result in suboptimal SFC placement, leading to resource wastage and/or SLA violations. Inherently, ML models lack robustness against adversarial attempts. Adversarial learning addresses this concern by leveraging carefully crafted adversarial (i.e. fake) samples, with minor perturbations to regular inputs [63, 64]. These can be used to inculcate robustness into ML models against data poisoning attacks. GANs have been widely used to generate such adversarial samples. GANs are a class of deep learning techniques that use two neural networks, discriminator and generator, to compete with each other for model training. However, GANs can suffer from training instability, due to fake training data that degrades model performance [65]. Therefore, ensuring convergence of GANs is an open research problem. Furthermore, the use of GANs to harden RL agents against complex threat vectors is rather unexplored. Adversarial deep RL with multi-agents [66, 67], trained across distributed virtualized environments, can also help alleviate the impact of adversarial attempts.

Bibliography

1 Anderson, T., Peterson, L., Shenker, S., and Turner, J. (2005). Overcoming the internet impasse through virtualization. *Computer* 38 (4): 34–41.

2 Turner, J.S. and Taylor, D.E. (2005). Diversifying the internet. *IEEE Global Telecommunications Conference (GLOBECOM)*, Volume 2, IEEE, p. 6.

3 Google IPv6 adoption statistics. https://www.google.com/intl/en/ipv6/statistics.html#tab=ipv6-adoption.

4 Liang, C. and Yu, F.R. (2014). Wireless network virtualization: a survey, some research issues and challenges. *IEEE Communication Surveys and Tutorials* 17 (1): 358–380.

5 Costa-Pérez, X., Swetina, J., Guo, T. et al. (2013). Radio access network virtualization for future mobile carrier networks. *IEEE Communications Magazine* 51 (7): 27–35.

6 Jinno, M., Takara, H., Yonenaga, K., and Hirano, A. (2013). Virtualization in optical networks from network level to hardware level. *Journal of Optical Communications and Networking* 5 (10): A46–A56.

7 Bari, Md.F., Boutaba, R., Esteves, R. et al. (2012). Data center network virtualization: a survey. *IEEE Communication Surveys and Tutorials* 15 (2): 909–928.

8 Jain, R. and Paul, S. (2013). Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine* 51 (11): 24–31.

9 Duan, Q., Yan, Y., and Vasilakos, A.V. (2012). A survey on service-oriented network virtualization toward convergence of networking and cloud computing. *IEEE Transactions on Network and Service Management* 9 (4): 373–392.

10 Drutskoy, D., Keller, E., and Rexford, J. (2012). Scalable network virtualization in software-defined networks. *IEEE Internet Computing* 17 (2): 20–27.

11 Blenk, A., Basta, A., Reisslein, M., and Kellerer, W. (2015). Survey on network virtualization hypervisors for software defined networking. *IEEE Communication Surveys and Tutorials* 18 (1): 655–685.

12 Alam, I., Sharif, K., Li, F. et al. (2020). A survey of network virtualization techniques for internet of things using SDN and NFV. *ACM Computing Surveys (CSUR)* 53 (2): 1–40.

13 Mijumbi, R., Serrat, J., Gorricho, J.-L. et al. (2015). Network function virtualization: state-of-the-art and research challenges. *IEEE Communication Surveys and Tutorials* 18 (1): 236–262.

14 Yi, B., Wang, X., Li, K. et al. (2018). A comprehensive survey of network function virtualization. *Computer Networks* 133: 212–262.

15 Ghaznavi, M., Shahriar, N., Kamali, S. et al. (2017). Distributed service function chaining. *IEEE Journal on Selected Areas in Communications* 35 (11): 2479–2489.

16 Carapinha, J. and Jiménez, J. (2009). Network virtualization: a view from the bottom. *ACM Workshop on Virtualized Infrastructure Systems and Architectures*, ACM, pp. 73–80.

17 Nikaein, N., Schiller, E., Favraud, R. et al. (2015). Network store: exploring slicing in future 5G networks. *International Workshop on Mobility in the Evolving Internet Architecture*, ACM, pp. 8–13.

18 Ayoubi, S., Limam, N., Salahuddin, M.A. et al. (2018). Machine learning for cognitive network management. *IEEE Communications Magazine* 56 (1): 158–165.

19 Boutaba, R., Salahuddin, M.A., Limam, N. et al. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications* 9 (1): 16.

20 Afolabi, I., Taleb, T., Samdanis, K. et al. (2018). Network slicing and softwarization: a survey on principles, enabling technologies, and solutions. *IEEE Communication Surveys and Tutorials* 20 (3): 2429–2453.

21 China Mobile Technology. (2016) Accelerating Business with OpenStack and OPNFV. https://object-storage-ca-ymq-1.vexxhost.net/swift/v1/6e4619c416ff4bd19e1c087f27a43eea/www-assets-prod/marketing/presentations/OpenStack-OPNFVDatasheet-A4.pdf (accessed 04 August 2020).

22 Yilma, G.M., Yousaf, Z.F., Sciancalepore, V., and Costa-Perez, X. (2020). Benchmarking open source NFV MANO systems: OSM and ONAP. *Computer Communications*. 161 86–98.

23 Rodriguez, V.Q., Guillemin, F., and Boubendir, A. (2020). 5G E2E network slicing management with ONAP. *Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, IEEE, pp. 87–94.

24 Rodriguez, V.Q., Guillemin, F., and Boubendir, A. (2020). Network slice management on top of ONAP. *IFIP/IEEE Network Operations and Management Symposium (NOMS)*, IEEE, pp. 1–2.

25 Karamichailidis, P., Choumas, K., and Korakis, T. (2019). Enabling multi-domain orchestration using Open Source MANO, OpenStack and OpenDaylight. *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, IEEE, pp. 1–6.

26 Cao, H., Hu, H., Qu, Z., and Yang, L. (2018). Heuristic solutions of virtual network embedding: a survey. *China Communications* 15 (3): 186–219.

27 Fischer, A., Botero, J.F., Beck, M.T. et al. (2013). Virtual network embedding: a survey. *IEEE Communication Surveys and Tutorials* 15 (4): 1888–1906.

28 Mijumbi, R., Gorricho, J., Serrat, J. et al. (2014). Design and evaluation of learning algorithms for dynamic resource management in virtual networks. *IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–9.

29 Mijumbi, R., Gorricho, J.-L., Serrat, J. et al. (2014). Neural network-based autonomous allocation of resources in virtual networks. *European Conference on Networks and Communications (EuCNC)*, IEEE, pp. 1–6.

30 Yao, H., Chen, X., Li, M. et al. (2018). A novel reinforcement learning algorithm for virtual network embedding. *Neurocomputing* 284: 1–9.

31 Yao, H., Zhang, B., Zhang, P. et al. (2018). RDAM: a reinforcement learning based dynamic attribute matrix representation for virtual network embedding. *IEEE Transactions on Emerging Topics in Computing* 1. https://ieeexplore.ieee.org/document/8469054

32 Yao, H., Ma, S., Wang, J. et al. (2020). A continuous-decision virtual network embedding scheme relying on reinforcement learning. *IEEE Transactions on Network and Service Management*. 17 (2): 864–875

33 Blenk, A., Kalmbach, P., Van Der Smagt, P. et al. (2016). Boost online virtual network embedding: using neural networks for admission control. *International Conference on Network and Service Management*, Montreal, Canada, October 2016, pp. 10–18.

34 Blenk, A., Kalmbach, P., Zerwas, J. et al. (2018). NeuroViNE: a neural preprocessor for your virtual network embedding algorithm. *IEEE International Conference on Computer Communications (INFOCOM)*, IEEE, pp. 405–413.

35 Yan, Z., Ge, J., Wu, Y. et al. (2020). Automatic virtual network embedding: a deep reinforcement learning approach with graph convolutional networks. *IEEE Journal on Selected Areas in Communications* 38 (6): 1040–1057.

36 Quang, P.T.A., Hadjadj-Aoul, Y., and Outtagarts, A. (2019). A deep reinforcement learning approach for VNF forwarding graph embedding. *IEEE Transactions on Network and Service Management* 16 (4): 1318–1331.

37 Solozabal, R., Ceberio, J., Sanchoyerto, A. et al. (2019). Virtual network function placement optimization with deep reinforcement learning. *IEEE Journal on Selected Areas in Communications* 38 (2): 292–303.

38 Xiao, Y., Zhang, Q., Liu, F. et al. (2019). NFVdeep: adaptive online service function chain deployment with deep reinforcement learning. *International Symposium on Quality of Service*, pp. 1–10.

39 Pei, J., Hong, P., Pan, M. et al. (2019). Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks. *IEEE Journal on Selected Areas in Communications* 38 (2): 263–278.

40 Zheng, J., Tian, C., Dai, H. et al. (2019). Optimizing NFV chain deployment in software-defined cellular core. *IEEE Journal on Selected Areas in Communications* 38 (2): 248–262.

41 Bunyakitanon, M., Vasilakos, X., Nejabati, R., and Simeonidou, D. (2020). End-to-end performance-based autonomous VNF placement with adopted reinforcement learning. *IEEE Transactions on Cognitive Communications and Networking*. 6 (2): 534–547

42 Pei, J., Hong, P., Xue, K., and Li, D. (2018). Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system. *IEEE Transactions on Parallel and Distributed Systems* 30 (10): 2179–2192.

43 Mijumbi, R., Hasija, S., Davy, S. et al. (2017). Topology-aware prediction of virtual network function resource requirements. *IEEE Transactions on Network and Service Management* 14 (1): 106–120.

44 Tang, P., Li, F., Zhou, W. et al. (2015). Efficient auto-scaling approach in the telco cloud using self-learning algorithm. *IEEE Global Communications Conference (GLOBECOM)*, IEEE, pp. 1–6.

45 Cao, L., Sharma, P., Fahmy, S., and Saxena, V. (2017). ENVI: elastic resource flexing for network function virtualization. *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud).*

46 Schneider, S.B., Satheeschandran, N.P., Peuster, M., and Karl, H. (2020). Machine learning for dynamic resource allocation in network function virtualization. *IEEE Conference on Network Softwarization (NetSoft).*

47 Shi, R., Zhang, J., Chu, W. et al. (2015). MDP and machine learning-based cost-optimization of dynamic resource allocation for network function virtualization. *IEEE International Conference on Services Computing*, IEEE, pp. 65–73.

48 Rahman, S., Ahmed, T., Huynh, M. et al. (2018). Auto-scaling VNFs using machine learning to improve QoS and reduce cost. *IEEE International Conference on Communications (ICC)*, IEEE, pp. 1–6.

49 Roig, J.S.P., Gutierrez-Estevez, D.M., and Gündüz, D. (2019). Management and orchestration of virtual network functions via deep reinforcement learning. *IEEE Journal on Selected Areas in Communications* 38 (2): 304–317.

50 Bega, D., Gramaglia, M., Banchs, A. et al. (2019). A machine learning approach to 5G infrastructure market optimization. *IEEE Transactions on Mobile Computing*. 19 (3): 498–512

51 Raza, M.R., Natalino, C., Wosinska, L., and Monti, P. (2019). Machine learning methods for slice admission in 5G networks. *OptoElectronics and*

Communications Conference (OECC) and 2019 International Conference on Photonics in Switching and Computing (PSC), IEEE, pp. 1–3.

52 Han, B., DeDomenico, A., Dandachi, G. et al. (2018). Admission and congestion control for 5G network slicing. *IEEE Conference on Standards for Communications and Networking (CSCN)*, IEEE, pp. 1–6.

53 Raza, M.R., Natalino, C., Öhlen, P. et al. (2019). Reinforcement learning for slicing in a 5G flexible RAN. *Journal of Lightwave Technology* 37 (20): 5161–5169.

54 Wang, X. and Zhang, T. (2019). Reinforcement learning based resource allocation for network slicing in 5G C-RAN. *Computing, Communications and IoT Applications (ComComAp)*, IEEE, pp. 106–111.

55 Koo, J., Mendiratta, V.B., Rahman, M.R., and Walid, A. (2019). Deep reinforcement learning for network slicing with heterogeneous resource requirements and time varying traffic dynamics. *arXiv preprint arXiv:1908.03242*.

56 Bega, D., Gramaglia, M., Fiore, M. et al. (2019). DeepCog: cognitive network management in sliced 5G networks with deep learning. *IEEE Conference on Computer Communications (IEEE INFOCOM)*, Paris, France, May 2019, pp. 280–288.

57 Li, R., Zhao, Z., Sun, Q. et al. (2018). Deep reinforcement learning for resource management in network slicing. *IEEE Access* 6: 74429–74441.

58 Hua, Y., Li, R., Zhao, Z. et al. (2019). GAN-powered deep distributional reinforcement learning for resource management in network slicing. *IEEE Journal on Selected Areas in Communications*. 38 (2): 334–349

59 Van Huynh, N., Hoang, D.T., Nguyen, D.N., and Dutkiewicz, E. (2019). Optimal and fast real-time resource slicing with deep dueling neural networks. *IEEE Journal on Selected Areas in Communications* 37 (6): 1455–1470.

60 Chowdhury, S.R., Bari, Md.F., Ahmed, R., and Boutaba, R. (2014). Payless: a low cost network monitoring framework for software defined networks. *IEEE Network Operations and Management Symposium (NOMS)*, IEEE, pp. 1–9.

61 Hadi, M., Pakravan, M.R., and Agrell, E. (2019). Dynamic resource allocation in metro elastic optical networks using Lyapunov drift optimization. *Journal of Optical Communications and Networking* 11 (6): 250–259.

62 Behzadan, V. and Munir, A. (2017). Vulnerability of deep reinforcement learning to policy induction attacks. *International Conference on Machine Learning and Data Mining in Pattern Recognition*, Springer, pp. 262–275.

63 Grosse, K., Papernot, N., Manoharan, P. et al. (2017). Adversarial examples for malware detection. In: *Computer Security – ESORICS 2017* (ed. S.N. Foley, D. Gollmann, and E. Snekkenes), 62–79. Springer International Publishing.

64 Pawlicki, M., Choraś, M., and Kozik, R. (2020). Defending network intrusion detection systems against adversarial evasion attacks. *Future Generation Computer Systems*. 110 148–154.

65 Kodali, N., Abernethy, J., Hays, J., and Kira, Z. (2017). On convergence and stability of gans. *arXiv preprint arXiv:1705.07215.*

66 Nguyen, T.T. and Reddi, V.J. (2019). Deep reinforcement learning for cyber security. *arXiv preprint arXiv:1906.05799*.

67 Zhang, K., Yang, Z., and Başar, T. (2019). Multi-agent reinforcement learning: a selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635.*