Traffic Engineering in Software-defined Networks using Reinforcement Learning: A Review

Delali Kwasi Dake[1], Griffith Selorm Klogo[3]
Henry Nunoo-Mensah[4]
Department of Computer Engineering, Kwame Nkrumah University of Science and Technology, Kumasi, Ghana

James Dzisi Gadze[2]
Department of Telecommunications Engineering
Kwame Nkrumah University of Science and Technology
Kumasi, Ghana

***Abstract*—With the exponential increase in connected devices and its accompanying complexities in network management, dynamic Traffic Engineering (TE) solutions in Software-Defined Networking (SDN) using Reinforcement Learning (RL) techniques has emerged in recent times. The SDN architecture empowers network operators to monitor network traffic with agility, flexibility, robustness and centralized control. The separation of the control and the forwarding plane in SDN has enabled the integration of RL agents in the networking architecture to enforce changes in traffic patterns during network congestions. This paper surveys major RL techniques adopted for efficient TE in SDN. We reviewed the use of RL agents in modelling TE policies for SDNs, with agents' actions on the environment guided by future rewards and a new state. We further looked at the SARL and MARL algorithms the RL agents deploy in forming policies for the environment. The paper finally looked at agents design architecture in SDN and possible research gaps.**

***Keywords*—Software defined networking; reinforcement learning; machine learning; traffic engineering**

I. Introduction

The emergence of fifth generation (5G) networks has propelled the growth of Internet of Things (IoT) in recent times. IoT is a rapid evolving technology that connects billions of devices to the internet [1]. With 5G, the rapid deployment of new and smart IoT applications are expected to reach 22.3 billion by 2024 and generate about 163 zettabyes (ZB) of data by 2025 [2] [3]. These new and dynamic applications are expected to benefit from the services 5G networks will provide: ultra-reliable and low latency communication (URLLC) [4], enhanced mobile broadband (eMBB) [5] and massive machine type communication, massive MIMO [6].

As shown in Fig. 1 and Fig. 2, the dynamic nature and requirements of IoT devices has necessitated a network deployment shift from the traditional networking architecture which are difficult to configure and manage to a more flexible programmable domain [7]. Software-Defined Networking (SDN) is a new networking paradigm that separates the data plane from the control plane [8] [9]. This separation makes the network more agile with centralized responsibility given to the controller [10]. The controller communicates with the application plane via the northbound APIs and the forwarding devices via the southbound APIs (OpenFlow). The automation and programmability of the SDN architecture helps to configure, secure, and optimize network resources [11] quickly whiles maintaining a good Quality of Service (QoS) [12] and Quality of Experience (QoE) [13].

Traffic Engineering (TE) in SDN involves the analysis of the networks state by the SDN controller to act on flow data through the rapid change in flow table information for forwarding devices [14]. Rerouting flows periodically to balance the loads on the network minimizes congestion and improves the overall network performance. A network experiences two kinds of traffic flows: elephant flows and mice flows [15]. The elephant flows are heavy traffic flows that requires more network resources whiles the rapid aggregation of the mice flows can equally degrade the network. These traffic flows continuously needs dynamic resource allocation for the efficient utilization of scarce network resources through TE.

With the advent of machine learning, port-based [16] and payload-based [17] flow classification techniques have become ineffective due to the dynamic port usage of IoT devices. The negative impact of packet out of order and packet loss in traditional TE techniques even worsens the case for the network operator.

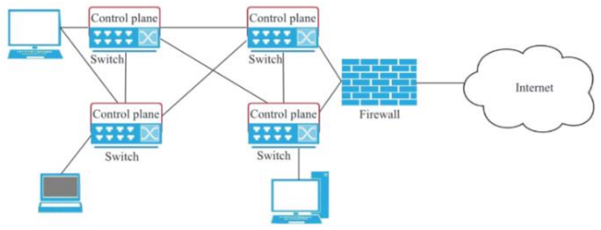

Fig. 1. Traditional Networking Architecture [7].

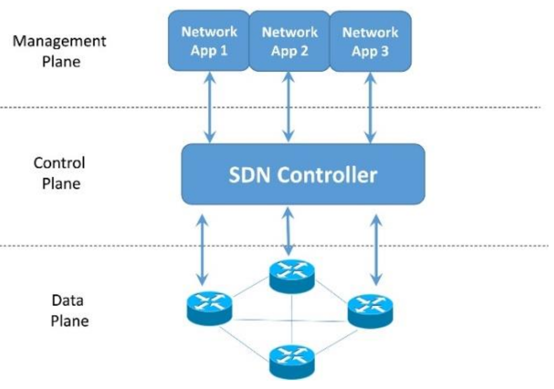

Fig. 2. Software-Defined Networking (SDN) Architecture [7].

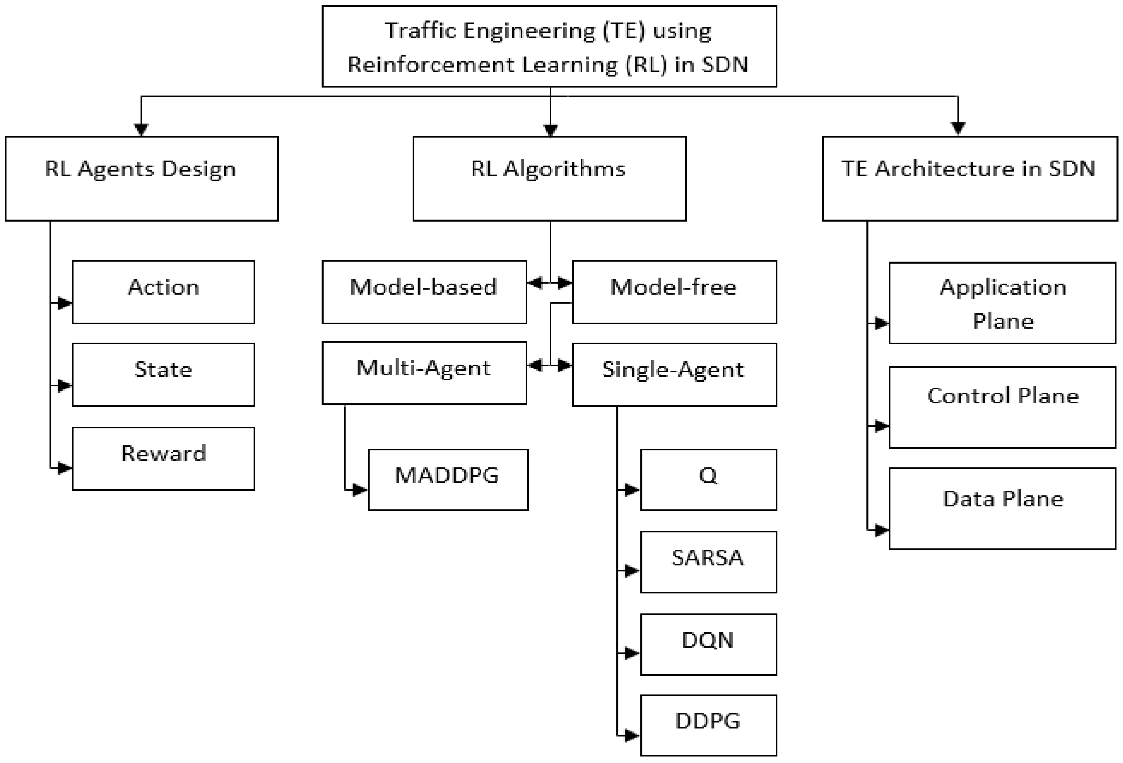

Fig. 3. TE using RL in SDN – Review Outline

Currently for TE, machine learning algorithms are adopted for intelligent flow re-routing with an efficient feature selection criterion [18] [19] in network flow analysis. Deploying these machine learning algorithms in the SDN controller will efficiently allocate network resources and formulate policies for optimal network performance with low overheads.

In this survey as outlined in Fig. 3, we reviewed popular Reinforcement Learning (RL) techniques used in SDN architecture for Traffic Engineering with limitations on parameters chosen and approaches for future research. The rest of the paper is organized as follows: Section II discusses the justification of RL for TE; Section III analysed the TE architecture integration in SDN based on policies and performance. Finally, Section IV looked at the research gaps identified from the survey.

II. Machine Learning with Reinforcement Algorithms

With the advent of Machine Learning [101] where automation modelling using data remains relevant, traditional algorithms [102][103][104] used in solving SDN-IoT related task is unfeasible. In supervised learning [103] agents are trained with a labeled dataset and later tasked to make predictions out of the learned data. Increased complexity in a dynamic environment with new IoT devices and variance in data will negatively affect the accuracy of supervised learning algorithms and predictions. Even worse is the time factor in retraining and relabeling of new data variance in an attempt to still adopt classification algorithms. With unsupervised learning [104] that uses unlabeled dataset, there is no guidance regarding the accuracy of the clustered dataset. Clustering algorithms alone is inefficient in an SDN-IoT environment that requires efficiency in diverse IoT applications. Reinforcement Learning (RL) defines the true automation of agents in an environment [20][105] with rewards as guidance on how well the agent is performing. Though complex, RL agents adapts to changing conditions in the environment by learning to solve tasks through trial-and-error approach. As the episodes progresses, agents adapt to successful actions through exploration and exploitation [106][107] on the stochastic environment. With the recent success of DeepMinds AlphaGo RL agent [108] that defeated the Go champion in 2016, the application dimensions of RL have become enormous. The only way packets can be routed intelligently in a network with varying and emerging IoT devices is to deploy RL agents to learn varying network state patterns with no exclusive data labels but with policies and actions.

III. TE using Reinforecement Learning in sdn

Reinforcement Learning (RL) is an area of machine learning where an agent is modeled to take sequence of actions informed by policies [20]. As shown in Fig. 4, the agent learns in an interactive environment and receives a reward through its actions [21, 22]. The set of actions presents a new state with corresponding reward to the agent. Unlike supervised learning [23] where a set of correct actions are provided as feedback to the agent, RL uses rewards and punishment as signals for positive and negative decisions. The goal is to use trial and error methods in getting positive rewards or build a suitable model that will maximize cumulative rewards for the RL agent.

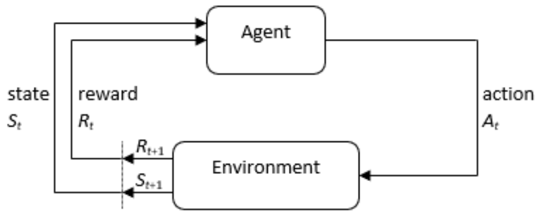

Fig. 4. Reinforcement Learning.

SDN provides centralised control with a unique advantage for intelligent TE framework implementation using RL. Network policies can easily be generated from the centralized control with corresponding TE rules to forwarding devices. With RL, the modelling of the agent's action on the environment with rewards fits into the network architecture of SDN and this expedites network control and management.

A. RL Agents Design

This section details the mathematical modelling of the state space with respect to actions and rewards. Agent design requires the environment to be monitored.

An agent based on the monitored metrics takes actions informed by policy decisions with a new state and a corresponding reward in guiding the next policy.

1) Action-State-Reward: RL agents are implemented in RL frameworks and modelled in SDN to learn critical network packet flow policies and provide routing solutions to forwarding devices. The agent takes an action on the environment and evaluates the actions based on rewards. Using its policy π , the agent performs an action *a*, which alters the environment state *s to s'*[24]. Based on the reward *r*, the agents policy is updated. In arriving at optimal policy, RL agents use Markov Decision Process (MDP) [25] to model actions on the environment with corresponding rewards. MDP is an intuitive and fundamental formalism for decision-theoretic planning (DTP) [26] and RL in stochastic domains. The MDPs have become the de facto standard formalism for learning sequential decision control problems [27].

Algorithm 1 Markov Decision Process (MDP)

```
An MDP is a 5-tuple $(S, A, P, R, \gamma)$, where;
    $S$ is a set of states
    $A$ is a set of actions
    $P\ (s, a, s')$ is the probability that action $a$ in state $s$ at time $t$
    will lead to state $s'$ at time $t + 1$
    $R\ (s, a, s')$ is the immediate reward received after a transition
    from state $s$ to $s'$, due to action $a$
        $\gamma$ is the discounted factor which is used to generate a discounted
        reward
```

For TE in SDN, RL agents are implemented differently based on the agents policy and the metrics for measuring TE success. The actions of the agents on the environment are rated by the rewards associated with it as the episode progresses.

CFR-RL agent [28]

State Space
$$s_t = TM_t \tag{1}$$

Action Space
$$\{0, 1, \dots, (N * (N - 1)) - 1\} \tag{2}$$

Reward Function
$$r = {}^{1}/_{U} \tag{3}$$

The CFR-RL agent resides in the controller of the SDN architecture. The RL agent uses a traffic matrix that contains the traffic demand of each flow as state. The objective is to avoid packet link congestion. As shown in equations 1 - 3, the CFR-RL agent samples K critical flows for a given stage s_t within N nodes. The CFR-RL agent then reroutes these critical flows and obtains maximum value in link utilization U as reward.

Q-DATA RL agent [29]

State Space
$$S_i \triangleq \{(f_i, \Delta f_i) : 0 < f_i \le f_{capi}; -f_{capi} \le \Delta f_i \le f_{capi}\} \tag{4}$$

Action Space
$$A_i \triangleq \{a : a \in \mathcal{F}\} \tag{5}$$

Reward Function
$$\begin{cases} \frac{\sum_{x=1}^{f_i} \Theta_x}{f_i}, & 0 < f_i < f_{capi}, \\ 0, & f_i = f_{capi}, \end{cases} \tag{6}$$

The Q-DATA agent resides in the application plane of SDN. As shown in equations 4 - 6, the Q-DATA RL agent has a defined state space where f_i is the current total number of flow entries in switch *i*; Δf_i is the number of flow entry changes between two consecutive observations and f_{capi} is the maximum number of flow entries in switch *i*. $(f_i, \Delta f_i)$ represents the state of an SDN switch *i*, as a tuple. For its action space, a represents a traffic flow matching scheme change related to a destination host and $\mathcal{F}$ denotes a list of all feasible match field combinations. With the reward function, f_i is the current total number of flow entries in the switch *i*; and Θ_x is an integer number representing the number of enabled match fields in flow entry *x*. An action has no reward if that action leads to the total number of current flow entries in the SDN switch *i* reaching the limit f_{capi}.

Mu [30]

State Space
$$s_i = (flow\_freq_i, flow\_recentness_i) \tag{7}$$

Action Space
$$a_{flow\_freq_i}^{increase} = (flow_{freq_i}{}^{increase}, flow_{recentness_i}) \tag{8}$$

Reward Function
$$r_t = Compare(overhead_{current_{best}}, overhead_t) \tag{9}$$

In [30], the RL agent resides in the controller of the SDN. As shown in equations 7-9, $flow\_freq_i$ represents the frequency of matched flows and $flow\_recentness_i$, an indication of flow duration in the memory of the switch. These are defined for the state space. The action space denotes an increase action on the flow frequency parameters. With the reward function the $overhead_{current_{best}}$ denotes the current best network control overhead obtained. A configuration with less overhead returns a positive reward, 1 to the RL agent otherwise a negative value -1 is returned. If $overhead_{current_{best}}$ and $overhead_t$ are equal, a reward value of 0 is given.

Huang [31]

State Space
$$s_f = (bandwidth_f, jitter_f, packet\_loss\_rate_f) \tag{10}$$

Action Space $$a_s = action_{bandwidth_f}^{routing\_path} \quad (11)$$

Reward Function $$r_s = \{MOS_{customer}, QoE_{customer}\} \quad (12)$$

With [31], the objective of the RL agent is to maximize the cumulative QoE of customers by dynamically allocation traffic in a multimedia environment. The RL agent resides in the controller of the SDN architecture. As shown in equations 10 – 12, the state of the environment refers to the state of flows and covers the following metrics: allocated bandwidth, the delay, the jitter and the packet loss rate of flows. The action includes: the path chosen (routing path) and the bandwidth adaptation of flows. The mean opinion score (MOS) [32] used to evaluate the QoE represents the reward function. A multi-layer deep neural network (DNN) is used to map the network and application metrics to the MOS.

Choi [59]

State Space $$s_i = (sampling\_period_i) \quad (13)$$

Action Space $$a_i^{increase} = (\, sampling\_period_{increase}) \quad (14)$$

Reward Function $$r_t = \begin{cases} 1, & hit\_ratio_{t-1} < hit\_ratio_t \\ 0, & hit\_ratio_{t-1} = hit\_ratio_t \\ -1, & hit\_ratio_{t-1} > hit\_ratio_t \end{cases} \quad (15)$$

In [59] RL framework is modelled to minimize the number of overflow occurrences. As shown in equations 13 - 15, the state space represents the size of the sampling period with a unit size of 500 ms. This ranges to 5,000 ms with a total of 10 states. The action space has three options: (i) increase sample period by unit size; (ii) decrease sampling period by unit size; (iii) maintain the sample period. Based on the percentage of table hits, three rewards are given. A reward of 1 is given when the measured hit rate is higher than the hit rate pre-action. If low, a reward of -1 is assigned. A reward of 0 is assigned if there is no change in the hit rate.

Fu [71]

State Space $$State = \left\{s = \left[FT_{sw_i,t_j}, PS_{pk,sw_i,t_j}\right] \middle| \, i \in [1,n], j \in [1,m], k \in [1,z]\right\} \quad (16)$$

Action Space $$Action = \{a_{p1}, a_{p2}, \dots \, a_{pk} \dots a_{pN}\} \quad (17)$$

Reward Function $$R_{elephant} = \alpha * (1 - PLR) + \beta * TP$$
$$R_{mice} = \lambda * (1 - PLR2) + \mu * (1 - DL) \quad (18)$$

In [71], flow table state and port state are responsible for collecting network statistics. The channels of the network represent the flow table utilization and its respective port rate of switches at current and previous states. For the state space modelling n, m and z respectively identifies the number of switches, moments and ports of a single switch. FT_{sw_i,t_j} represents the flow table utilization rate of switch i at the moment t_j and ranges from 0 to 1. PS_{pk,sw_i,t_j} represents the port rate of port k in switch i at the moment t_j. The action space comprises of p_1 *to* p_N which indicates all paths in the network, $a_{pk} \in \{0,1\}$. If $a_{pk} = 1$, the current flow is assigned to path k else $a_{pk} = 0$. For the reward function, the elephant-flows PLR, represents the average packet loss rate of elephant-flows in the network, TP is the average throughput of elephant-flows after processing. α and β are the weights of the PLR and TP respectively. With the mice-flows, $PLR2$ indicates the average packet loss rate of mice-flows and DL represents the normalized average delay. λ and μ identifies the weight of the $PLR2$ and DL , respectively.

Zhang [86]

State Space $$s = (nc, src, dst, avail) \quad (19)$$

Action Space $$a = (a_1, , a_i \dots, a_j, path) \quad (20)$$

Reward Function $$r = \frac{1}{L}\sum_{l=1}^{L}(2\frac{bl}{bw_l} - 1) - \beta\frac{2}{\pi}\arctan(\sigma) + 1 \quad (21)$$

In [86] the state comprises of four components; name of the requested content, source, destination and available link bandwidth.

With the action, a_i denotes the ith destination node split ratio and relates to the content request sent to that destination node using selected transmission links. The reward is meant to improve load balance and throughput. The $\frac{1}{L}\sum_{l=1}^{L}(2\frac{bl}{bw_l} - 1)$ reveals throughput impact in relation to available normalized bandwidth. The - $(\frac{2}{\pi})\arctan(\sigma) + 1$ indicates the load balance with normalized variance of available bandwidth. A value close to 1 signals a preferred action with a reverse value close to -1, a penalty. β = 1 is a factor used to balance the throughput and the load balance.

B. RL Algorithms

In this section, we reviewed the algorithms the RL agents use to formulate policies that informs the action taken by the agent on the environment as the episode progresses. For effective TE and policy enforcement on the environment, RL agents learns to take the best actions for traffic optimization in respect to cumulative future rewards. RL algorithms are distinguished into two main classes: the model-free (direct) and model-based (indirect) methods [33, 34, 35].

1) Model-based RL methods: Model-based RL algorithms utilizes a model when the RL agent interacts with the environment. The model keeps track of transition dynamics of the network to derive optimal actions and rewards [35]. When the model is referenced, the RL agent can make predictions about the next state and reward before an action is taken. Model-based RL methods are data efficient but struggles to achieve asymptotic performance for real-world applications [36]. For model-based RL methods, the interaction between the RL agent and the environment is modeled as a discrete-time Markov Decision Process (MDP) $\mathcal{M}$ and defined by the tuple [36]:

$(S, A, p, r, \gamma, p_0, H)$. Where S is the set of states, A the action space, $p(s_{t+1}| \, s_t, a_t)$ the transition distribution, $r : S \times A \rightarrow \mathbb{R}$ as a reward function, $p_0 : S \rightarrow \mathbb{R}_+$ represents the initial state distribution, γ the discount factor, and H the horizon of the process. The return function is defined as the sum of rewards $r\,(s_t, a_t)$ along a trajectory τ: $=(s_0, a_0, \dots, s_{H-1}, a_{H-1}, s_H\,)$. The goal of the reinforcement learning is to find a policy π: $S \times A \rightarrow \mathbb{R}^+$ that maximizes the expected return. The model-based learns the transition distribution from the observed transitions using parametric approximator $\acute{p}_{ø}(s'|s,a)$. The parameter ø of the dynamic model are optimized to maximize the log-likelihood of the state transition distribution. Though model-based RL methods are data efficient, they have high computational complexity and the degree of potential error in maximizing a reward is compounded..

2) Model-free RL methods: Model-free RL algorithms do not utilize a model and thus the rewards and the optimal actions are derived through trial-and-error approach with the environment [37]. These set of algorithms operate over an unordered list of actions, with a positive or negative reward value. The RL agents that utilizes model-free algorithms increases the value associated with a positive action which helps the agent to learn from direct experience. Agents in model-free RL are represented with policy optimization and Q-learning approaches [38]. With policy optimization, the agents learns directly the policy function that maps state to action without a value function. The Q-learning approach learns the action-value function $Q(s,a)$; how good to take an action at a particular state. A scalar value is assigned over an action *a*, given the state *s* [39]. Model-free RL methods have low computational complexity but more data dependent. For TE, model-free RL methods are frequently used for RL agent sequencing and to implement policies on the environment.

3) Single Agent Reinforcement Learning (SARL): In a SARL, there is only one agent that interacts with the environment to maximize rewards. The SARL implementation is suitable for simple network management with slower convergence and learning experience. The SARL implemented algorithms are either value-based, policy-based or both [48]. As shown in Fig. 5, the SARL through the SDN controller collects information from the environment through the forwarding devices.

The agent upon receiving the state information performs a set of actions on the environment through the SDN controller. These actions are guided by policy algorithms. The episode results in a new state and rewards.

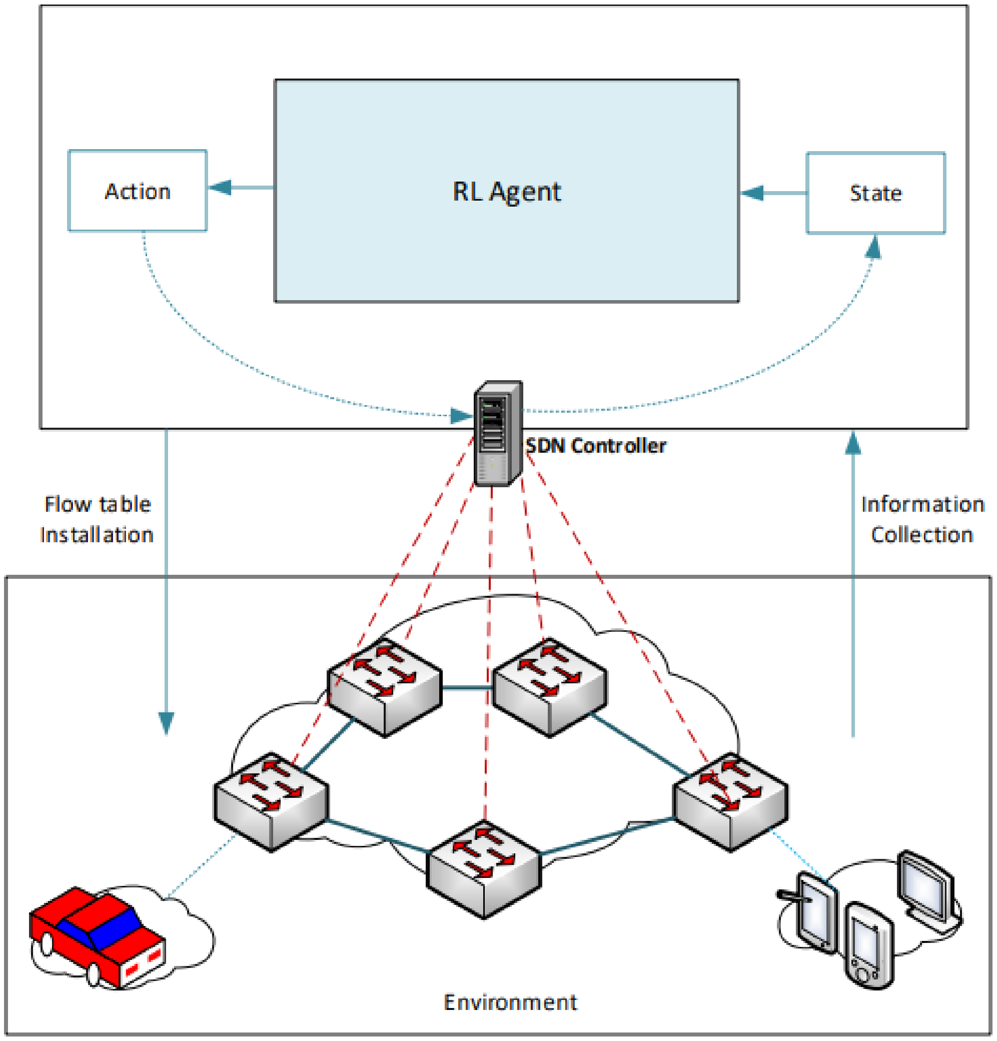

Fig. 5. SARL.

a) Q-learning Algorithm: Q-learning [40] is an off-policy, value-based algorithm that takes a random actions based on the $\epsilon - greedy$ policy, where the probability of a random decision is determined by the value of epsilon ϵ. During the learning phase, the Q-learning agent initializes the Q-table for all state-action pairs and updates it using:

$$Q\_(t+1)\,(s\_t, a\_t\,) = Q(s\_t, a\_t\,) + \alpha[\mathcal{R}\_i\,(s\_t, a\_t\,) + \gamma max Q\_t\,(s\_(t+1), a\,) - Q\_t\,(s\_t, a\_t\,)] \quad (22)$$

The Q-learning agent generates the optimal policy $\pi^*(s)$ for a state s representing an action a that needs to be taken to maximize the value of the $Q_*(s,a)$ function, $\pi^*(s) = arg\,max_a\, Q_*(s,a)$.

Algorithm 2 Q-learning [40]

```
1:  Inputs: $\mathcal{F}$; for a state-action pair $(s,a)\ \forall s \in S_i, a \in \mathcal{A}_i$,
    initialize a $Q$-table entry arbitrarily; initialize values of $\alpha$, $\gamma$ and $\epsilon$,
    respectively.
2:  loop
3:      Current state $s_t$.
4:      Executive action $a_t$ according to an exploratory policy ($\epsilon$).
5:      Obtain a new state $s_{t+1}$ and an immediate reward $\mathcal{R}_i$.
6:      Update the $Q$-table entry for $Q(s_t, a_t)$.
7:      Update $s_t \longleftarrow s_{t+1}$.
8:  end loop
9:  Outputs $\pi^*(s) =$ arg $max_a\, Q_*(s,a)$.
```

Phan *et al.* [29] proposed the Q-learning algorithm in maximizing traffic flow monitoring in SDN switches. It embeds a Support Vector Machine (SVM) [49] algorithm in the application plane of the SDN architecture to predict the performance degradation of the switches as the episode progresses. To reduce the long-term control plane overhead capacity limitation of Ternary Content Addressable Memory (TCAM) in OpenFlow switches, [30] proposed a Q-learning algorithm for SDN flow entry management. The framework determines the forwarding rules that remains in the flow table of the SDN switches and those processed by the controller in case of a table-miss on the switches. In [50] a Q-learning algorithm is proposed to reduce the latencies and improve the bandwidth utilization in the UbuntuNet Alliance National Research and Education Network (NRENs) SDN switches. The proposed framework adapts forwarding devices by learning from experience using multipath propagation. In dealing with bandwidth overhead caused by Dijkstra's shortest path first module [51] in an OpenDayLight (ODL) architecture meant for efficient packets delivery, [52] proposed a congestion prevention mechanism using Q-learning in SDN. With [52], the set threshold values are defined in SDN controllers to enable threshold bandwidth detections. The optimal path chosen is delivered to the OpenVSwtiches (OVS) after Q-routing by the controller during network congestion. To balance the network load in SDN, [53] proposed a Q-learning approach to reduce the number of unsatisfied users in a 5G network architecture. The researchers used a flow admission control technique with a fairness function to enhance the per-flow resource allocation in the network. In [54] a load balancing architecture is proposed for SDN networks that uses supervised Bayesian Network (BN) to solve the problem of Q value local maximum [55] in a Q-learning RL algorithm. The combination of the BN in Q-learning helps the controller select the most optimal strategy for network load balancing during congestion. For TE load balancing optimization in master controllers, [56] proposed a dynamic switch migration algorithm to slave controllers using Q-learning in SDN. The switch migration problem (SMP) is modeled and used to redefine the Q-learning parameters. The Q-learning is then used to learn the current status of SDN to select the best switches for load migration. For an efficient path selection technique in load balancing, [57] proposed a Q-learning algorithm for path selection and flow forecasting [58]. It has an integrated centre that uses Deep Neural Networks (DNNs) to process uncertain network traffic and uses Q-learning to resolve the optimal path based on the results of the DNN. The DNN path selection are obtained from the bandwidth utilization ratio, packet loss rate and transmission latency which forms the inputs to the DNN. The output which is fed into the Q-learning is derived from the corresponding link score. For timely eviction of inactive flow entries and to avoid overflows in the memory of SDN switches, [59] proposed a Q-learning User Datagram Protocol (UDP) [60] flow eviction strategy for UDP flows. The Q-learning is used to dynamically resize the sampling period as the most critical parameter in the RL architecture. This advertently maximizes the table hit rates of the UDP flows in the SDN.

b) State-Action-Reward-State-Action (SARSA): SARSA [61] is an on-policy algorithm which uses the action performed by the current policy to learn the Q-value. As shown in Eq. 23 [61] and Eq. 24 [40], the update rule for SARSA varies from that of Q-learning algorithm in the execution of actions. In SARSA, update estimates are based on the same action taken whiles in Q-learning, the update estimates are based on the number of possible actions that maximizes the post-state Q function, $Q(s_{t+1}, a')$.

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (23)$$

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max Q(s_{t+1}, a') - Q(s_t, a_t)] \quad (24)$$

For dynamic load balancing in multiple controllers due to switch migration conflicts, [62] proposed a SARSA-Bayesian RL algorithm for a multi-controller cluster design in SDN. With knowledge of the real-time load and controller's communication consumption, a request response model using the Bayesian [63] algorithm is combined with the SARSA RL mini-framework in a switch migration technique to the lighter controller. For a multi-layer hierarchical SDN to be effective in handling traffic, [64] proposed the SARSA algorithm for QoS provisioning. With each pre-flow, the switch contacts the SDN controller. The controller uses the SARSA algorithm to implicitly detect the QoS requirement of each flow and computes the corresponding optimum traffic path based on the needed QoS requirement. The next hop in the switch forms the basis for the next action from the source to the destination switch. To convey a massive IoT data through a limited bandwidth efficiently [65] proposed a SARSA algorithm for resource allocation through cognitive communications in the SDN-enabled environment. The SARSA agent communication is modelled with a buffer metric that manages the aggregator's output queue transmissions and reflects dynamically in the IoT data demands. This modification targeted at Publish/Subscribe (Pub/Sub) paradigms preserves the Pub/Sub bandwidth with less computational resources. In order to adapt VS-routing [67] optimization to SDN networks, [66] proposed a network hop count technique to improve the reward function of SARSA algorithm. The VS-routing introduces an $\varepsilon - Greedy$ function in the network hop count which is calculated to select the optimal route and avoid the long package queue of network links in the SDN architecture.

c) Deep Q-Network Algorithm (DQN): With the advent of Artificial Neural Networks, (ANNs) a class of RL agents that utilizes Q-learning with Deep Neural Networks (DNNs) [41] in discrete domains for TE has emerged. DQN uses feedforward neural networks with three components: (i) Neurons that are interconnected using direct links to form a network, (ii) Weights associated with each connection, (iii) Layers consisting of a number of neurons and multiple hidden layers.

Algorithm 3 Deep Q-Network Algorithm [41]

```
Pre-condition:
        Initialize experience memory $M$
        Initialize action-value pair $Q$ with random weights
        Initialize state $s_t$
        Initialize goal $\mu$
Procedure:
1:      improvement = 0
2:      repeat
3:          for (step = 0; step < learning_iteration; step++)
4:              Get action $a_t$ from $s_t$ using $\epsilon - greedy$ policy
5:              Get parameter $param_t$ from $s_t$ using $\epsilon - greedy$ policy
6:              $\epsilon = \epsilon -$ (step / learning_iteration)* $\epsilon$
7:              Take action $a_t$ on $param_t$ and receive reward $r$, control
                overhead $c_t$
8:              Observe new state $s_{t+1}$
9:              Store experienced memory $(s, a, r, s_{t+1})$ into $M$
10:             Sample $n$ random transitions $(s', a', r', s'')$ from $M$
11:             Update $transition\ tt \leftarrow r' + \gamma * \max(s'' - a'')$
12:             Update the $param_t$ of $\theta_i$
13:             Train the $Q$ network using $loss = (tt - Q(s', a'))^2$
14:             $improvement$ = get_improvement ( $best_t, worst_t$)
15:         end for
16:     until improvement > $\mu$
```

The DQN has an experience memory for storing experienced transitions (s, a, r, s') unlike the Q-learning. The discount factor γ and the state of the Q-Network in the ith iteration, θ_i are used to update the experienced transitions with a training principle using a loss function. The $\epsilon - greedy$ policy helps select the action based on the highest Q-value associated with that action after the training. For the RL agent to choose random actions, the ϵ value is set to 1 at the start of the learning process but decreased over time in order to maintain a fixed exploration rate. The DQN keeps track of the chosen parameter corresponding to the Q-value of each action with a terminal, μ..

In [28], the DQN is used to learn a policy to select critical flows based on a given traffic matrix. The Critical Flow Re-routing-Reinforcement Learning (CFR-RL) agent then reroutes the selected flows for a balanced link utilization using Linear Programming (LP). For an efficient SDN flow entry level management with a TCAM enabled OpenFlow switches [30] proposed a DQN algorithm to obtain the flow entries and reduce the long-term control plane overhead between the SDN switch and the controller. The DQN agent automatically finds the values of decision parameters that effectively selects the candidates rule in the switch's flow table for a higher table-hit rate. For flexible network management through TE, [68] proposed a DQN based dynamic controller placement caused by flow fluctuations in SDN. The D4CPP agent in [68] integrates historical network data into the controller deployment. The real-time switch-controller mapping decisions is then triggered with inherent adaptation to the dynamic flow fluctuations in the network. For effective TE among distributed controllers in SDN, [69] proposed a DQN based switch and controller selection scheme for switch migration and switch-aware reinforcement learning-based load balancing (SAR-LB). The SAR-LB adopts the utilization ratio of diverse resource types in both controllers and switches as inputs to the neural network for a dynamic load distribution among the controllers in the network. Yao *et al.* [70] proposed a DQN-based energy-efficient routing solution for full load software-defined data centers. The optimization is for the DQN to find energy-efficient routing paths and load-balancing between controllers in reducing energy consumption in the network. The enhanced DQN-based energy-efficient routing (DQN-EER) algorithm learns directly from experience. At the same coordinated time, it selects the arriving flows and the energy-saving control path at the in-band control mode whiles detecting the energy-saving routes for the data center. Fu *et al.* [71] proposed the detection of mice and elephant flows in an SDN-enabled data center using two DQNs. The DQNs are built and trained to generate efficient routing strategies using convolutional neural networks (CNNs) [72][73] to avoid possible network congestion. For efficient latency management in SDN, [74] proposed a DQN agent that inherently predicts optimal traffic paths and future traffic demands through the SDN switches. Whiles formulating the flow rules placement policy as an Integer Linear Program (ILP), [74] used a traffic prediction module with a long short-term memory (LSTM) [75][76] neural networks algorithm. To further minimize network delay, a proposed DQN-TP (traffic prediction)-based heuristics defect-tolerant routing (DTR) [77] algorithm interacts dynamically with the DQN agent module in the controller of the SDN architecture.

d) Deep Deterministic Policy Gradient (DDPG): In combining policy gradient and Q-learning, Deep Deterministic Policy Gradient (DDPG) [42][79] is used as an off-policy, actor-critic technique consisting of two modes; actor and critic as shown in Fig. 6. The actor is the policy network and the critic, the Q-value for training the actor network.

Algorithm 4 Deep Deterministic Policy Gradient (DDPG) Algorithm [42]

```
1:  Input: Initial policy parameters $\theta$, Q-function parameters $\phi$, empty replay buffer $\mathcal{D}$
2:  Set target parameters equal to main parameters $\theta_{targ} \longleftarrow \theta, \phi_{targ} \longleftarrow \phi$
3:  repeat
4:      Observe state $s$ and select action $a = \text{clip}\,(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
5:      Executive $a$ in the environment
6:      Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
7:      Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
8:      If $s'$ is terminal, reset environment state
9:      if it's time to update then
10:         for however many updates do
11:             Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:             Compute targets
                    $y(r, s', d) = r + \gamma(1-d)Q_{\phi\text{targ}}(s', \mu_{\theta\text{targ}}(s'))$
13:             Update Q-function by one step of gradient descent using
                    $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} (Q_\phi(s,a) - y(r\, s', d))^2$
14:             Update policy by one step of gradient ascent using
                    $\nabla_\phi \frac{1}{|B|} \sum_{s \in B} (Q_\phi(s, \mu_\theta(s))$
15:             Update target networks with
                    $\theta_{\text{targ}} \longleftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$
                    $\theta_{\text{targ}} \longleftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta$
16:         end for
17:     end if
18: until convergence
```

DDPG uses DQNs replay buffer to gather offline unrelated experiences obtained by the agents whiles performing actions on the environment. At each time step, the actor and the critic are updated by uniformly sampling a minibatch from the replay buffer. DDPG uses soft target, θ_{targ} updates rather than directly copying the weights to the target network. DDPG further utilizes batch normalization which helps normalize each dimension across the samples in a mini-batch to have unit mean and variance. DDPG algorithm is suitable for continuous action space and state representations.

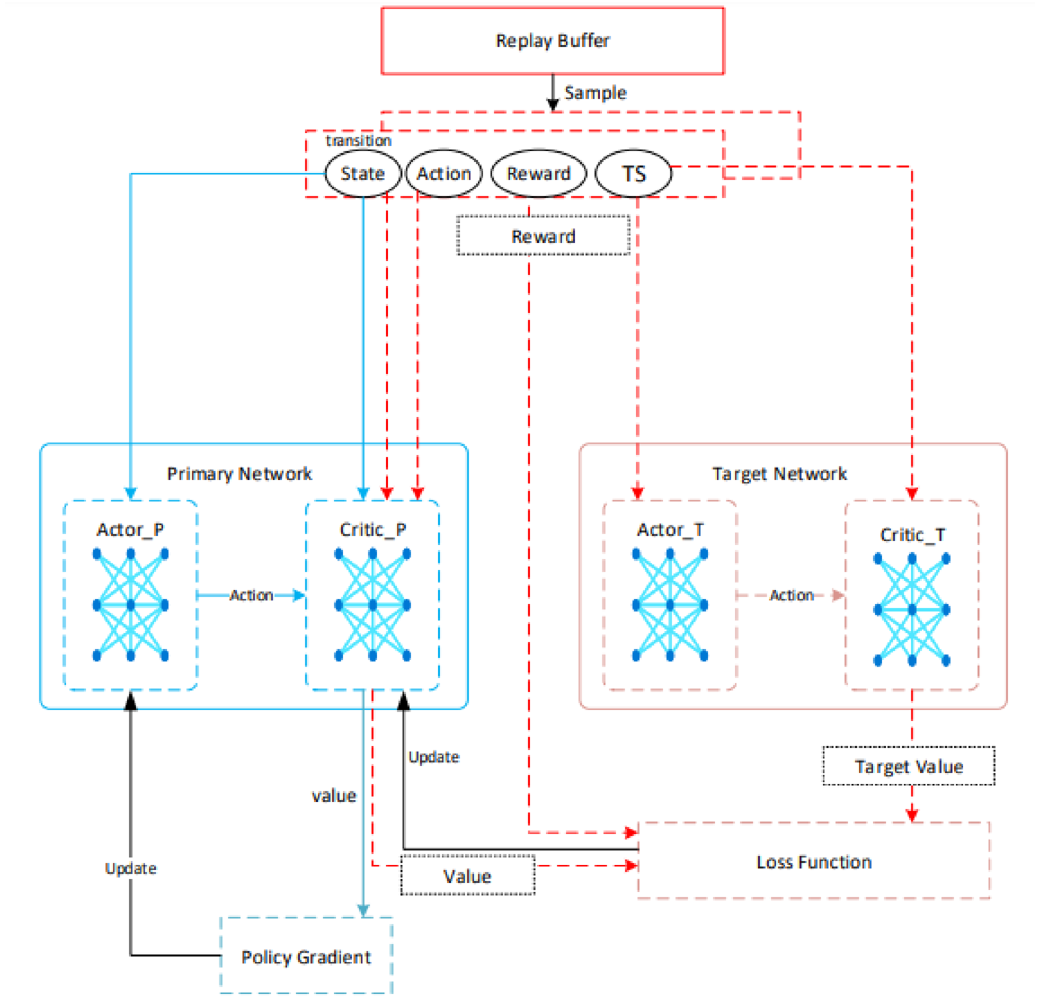

Fig. 6. Actor-Critic Model of DDPG [31].

In [31], the DDPG algorithm is used for multimedia traffic control with the objective of maximizing cumulative Quality of Experience (QoE) for network users. The DDPG agent enforces bandwidth adaptation and path chosen actions for all multimedia flows in the SDN-enabled environment. To maximize the QoE for users, a multi-layer deep neural network is used to map the network and application metrics to the mean opinion score (MOS) [78] obtained from users. Stampa *et al.* [80] proposed a DDPG agent for dynamic routing in SDN. The architecture embeds an integrated fully-connected feed-forward neural network (FFNN) [81] in the framework to re-define the feature extraction of the actor-critic network. To improve the learning rate of DDPG for effective routing optimization, [82] proposed a dynamic planning of the experience pool capacity with respect to the current iteration number. This accelerates the growth rate of the previous pool by reducing its capacity in affecting subsequent learning rates. In [83] a deep-reinforcement-learning-based quality-of-service (QoS)-aware secure routing protocol (DQSP) is proposed using DDPG algorithm. The DQSP adds an intelligent layer above the control layer which generates the routing policy and evaluates the network performance through the rewards obtained by the DDPG policy. The DQSP protocol guards against gray hole attack [84] and DDoS [85] whiles ensuring an efficient routing planning through the environment-aware module of the control layer. Zhang *et al.* [86] proposed a DDPG-based intelligent content-aware TE (iTE) which leverages on information centric networking (ICN) [87] to optimize traffic distribution in SDN. The DDPG agent together with other TE algorithms are embedded in a parallel decision-making (PDM) module in the controller. This module receives the cache information and the link bandwidth from the switches to activate and update its neural networks with a reward feedback. In [88] a DDPG-based network scheduler for deadline-specific SDN heterogenous networks is proposed. The DDPG agent receives a deadline-ware data transfers from the SDN switches and schedules the flows by initializing a pacing rate at the source of the deadline flows. The actor-critic model in the DDPG agent handles larger and a more generalized scheduling problem that maximizes and assigns the aggregated utility value to each flow if the deadline is met. For intelligent routing in software-defined data-centers (SD-DCN), [89] proposed a deep reinforcement learning based routing (DRL-R) consisting of DDPG-DQN agent to perform a reasonable routing adapted to the network state. DRL-R agent efficiently allocates cache and bandwidth in the network to improve routing performance by delay reduction. This is done through the quantification of the overall contribution score in the network and a change in the routing metric from a single link state to the resource-combined state.

4) Multi-Agent Reinforcement Learning (MARL): In MARL systems, multiple agents collectively learn and collaborate in a deterministic or a stochastic environment [90, 91, 92]. Multi-agent systems are seen in domain applications including: network resource management, computer games, distributed networking, cloud computing and intrusion detection systems. Experience sharing and faster convergence has necessitated a shift in research direction from SARL to MARL in recent times. With a coordinated policy, multi-agents learn and optimize towards an accumulated global reward [93, 94] in the network framework. As a result, the dynamics in state transitions in MARL are dependent on the joint action of all active agents as shown in Fig. 7.

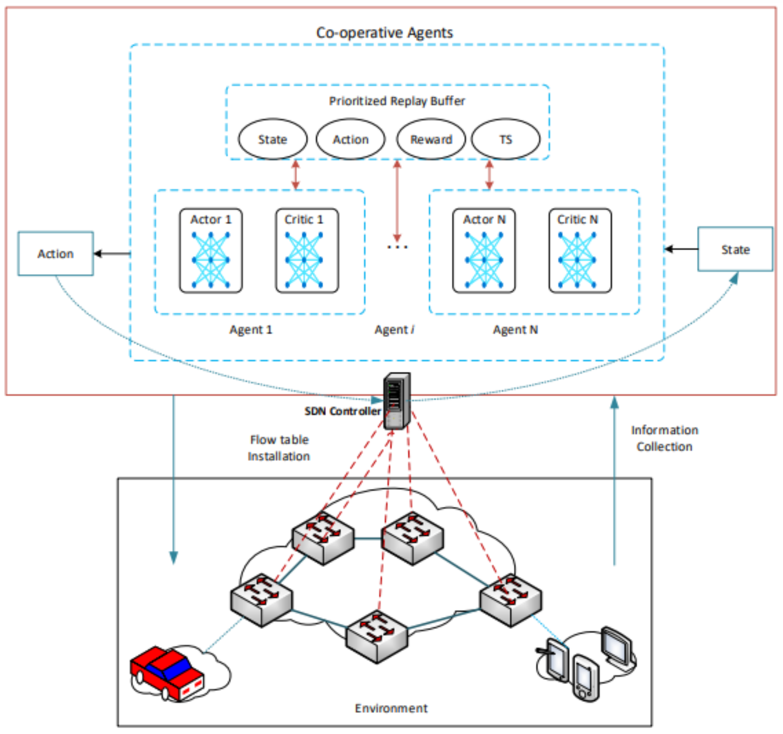

Fig. 7. MARL

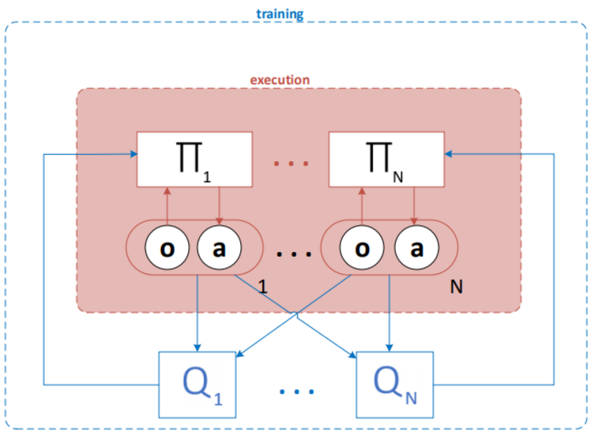

Fig. 8. MADDPG.

a) Multi-Agent Deep Deterministic Policy Gradient (MADDPG): MADDPG [95, 96, 97] is an actor-critic multi-agent extension of DDPG where the critic network is augmented with information from other agents in a decentralized execution. In MADDPG actor-critic architecture, each agent has its own actor and critic network. The critic network of each agent has full visibility of the actions and observation of other agents.

The actor network on the other hand only executes the action for its local agent given the state. In Fig. 8, the actor π_n takes an observation, o as state to give an action, a whiles the critic network, Q_n takes an observation and the action of the actor, to train the actor. The critic has dependent view from other critic networks whiles training the actor network.

Algorithm 5 Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [97]

```
1:  for episode = 1 to $M$ do
2:      Initialize a random process $\mathcal{N}$ for action exploration
3:      Receive initial state $x$
4:      for $t = 1$ to max-episode-length do
5:          for each agent $i$, select action $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}_t \; w.r.t.$ the current policy and exploration
6:          Executive actions $a = (a_1, \ldots, a_N$ and observe reward $r$ and new state $x'$
7:          Store $x, a, r, x'$ in replay buffer $\mathcal{D}$
8:          $x \leftarrow x'$
9:          for agent $i = 1$ to $N$ do
10:             Sample a random minibatch of $S$ samples $x^j, a^j, r^j, x'^j)$ from $\mathcal{D}$
11:             Set $y^j = r_i^j + \gamma Q_i^{\mu'}(x'^j, a'_1, \ldots, a'_N)|a'_k = \mu'_k(o_k^j)$
12:             Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j (y^j - Q_i^{\mu}(x^j, a_1^j, \ldots, a_N^j))^2$
13:             Update actor using the sampled policy gradient:
                $\nabla_{\theta i} J \approx \frac{1}{S}\sum_j \nabla_{\theta i}\mu_i(o_i^j)\nabla a_i Q_i^{\mu}(x^j, a_1^j, \ldots, a_N^j)|a_i = \mu_i(o_i^s)$
14:         end for
15:         Update target network parameters for each agent $i$
                $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
16:     end for
17: end for
```

In [98], a MADDPG-based traffic control and multi-channel reassignment (TCCA-MADDPG) algorithm is proposed for the core backbone network in SDN-IoT. The TCCA-MADDPG algorithm reduces the channel interference between links by considering the policies of other neighbouring agents using a cooperative multi-agent strategy. To maximize network throughput and minimize packet loss rate and time delay, the TCCA-MADDPG uses a joint traffic control mechanism modelled with a partially observable markov decision process (POMDP) to optimize traffic performance. Yuan *et al.*, [99] proposed a dynamic controller assignment using MADDPG for effective TE in Software Defined Internet of Vehicles (SD-IoV) [100]. For controllers to make local decision in coordination with neighboring controllers, a real-time distributed cooperative assignment approach is used via the actor-critic model of the MADDPG. To get a faster MARL global convergence whiles minimizing delay, a centralized training approach using global information to attain optimal local assignment is adopted in the model development.

C. TE Architecture in SDN

In this section, we looked at the design placement of the RL agents in the SDN architecture and the communication principles adopted with the controller. The architecture of RL systems varies based on the RL agent policy algorithms, the actions selected and the environment. The agent frameworks are designed to enhance positive rewards and proactively prevent network performance degradation through forwarding devices. Different components of the RL agents design in SDN are situated in the application plane, control plane and the data plane.

1) RL agent in control plane: For easier policy formulation and faster communication between the controller and RL agent, most TE SDN designs [28][30][31][52][54][57][70][86][89] situate the RL agent in the control plane of the SDN architecture. In [28], the CFR-RL agent resides in the controller and uses a neural network trained with reinforcement algorithm [43] to map a traffic matrix to a combination of critical flows. After training, the CFR-RL applies the critical flow selection policy to each real time traffic matrix provided by the controller. The SDN controller then reroutes the selected critical flows by installing and updating flow entries of the switches whiles the remaining flows continue the normal route using Equal-Cost Multi-Path (ECMP) [44] TE technique by default. In [30], the RL agent is deployed in the controller and utilizes the flow match frequency and the flow duration to determine the flow entries that should be kept on the switch. To maximize the long term reward, the RL agent lowers the configuration overhead and the number of table-miss events. To achieve the expected reward, the RL agent splits the pool of flow entries into two parts: the local switch entries and the remote controller entries. This will reduce the control plane overhead given the Ternary Content-Addressable Memory (TCAM) [45] size of the SDN switches. With [31] the RL agent is the controller and serves as the centralized control to collect stats, make decision and take actions. The state reflects the situation in the environment and covers metrics: allocation of bandwidth, delay, jitter and the packet loss rate of flows. The action involves the path chosen and the bandwidth adaption for multimedia flows. The reward is the QoE received from the environment. To evaluate the QoE, the multi-layer deep neural network is used to map the network and application metrics to MOS [46]. [52] also proposed the controller is the RL agent and programmed with the Q-learning algorithm to detect network congestion and find optimal path to be delivered to the OpenVSwitch (OVS). In [57] the control layer has an intelligent center connected to the SDN controller. For efficient load balancing, the intelligent center uses the Q-learning algorithm to find optimal paths and returns aggregated path routing decisions to the controller. The DQN-EER architecture [70] has the RL agent programmed in the SDN controller using the DQN algorithm. The DQN is modified with deep convolutional neural networks (CNNs), empirical replay to train the agent and independent target networks to train the primary critic network. In [86] the intelligent content-aware traffic engineering (iTE) RL agent is deployed in the controller of the SDN architecture. It received cache information from the ICN-enabled switches and uses parallel execution module embedded with multiple DRL-based TE algorithms to determine the best routing paths for the flows in the network.

2) RL agent in Application Plane: For easier system failure checks in SDN, [29] [71][83] TE frameworks situate the RL agent in the application plane. The Q-DATA [29] framework architecture has a built-in forwarding application located in the control plane and a Q-DATA application residing in the SDN application plane. Initially, the built-in forwarding application module is instructed by the Q-DATA application through a REST API to apply the Full Matching Scheme (FMS) strategy at the switches. The Q-DATA application has a statistics collector module which periodically collects raw information about traffic flows at the SDN switches from the SDN controller. The statistics is then forwarded to a statistics extractor and distributor module for extraction and distribution to other modules. The SVM based performance degradation prediction module anticipates the performance degradation of the SDN switches before it occurs and provides the prediction results to the Q-learning based traffic flow matching policy creation module and the MAC matching only scheme control module. The MAC matching only scheme control module monitors and checks conditions for a traffic flow matching scheme change to FMS in the SDN switches. In [71] the AI Plane is used as the Application Plane in the SDN architecture. The RL agent is embedded in the AI Plane and uses the DQN to learn the best optimal routing paths for the mice and elephant flows by obtaining the flow type, network state information and network performance evaluation from the control plane of the SDN architecture. In [83], the DQSP architecture has an agent layer that is embedded in the application layer of the SDN architecture. The DQSP agent through the controller is aware of the underlying network environment and generates routing policies for the controller to executive. It receives the reward evaluation and adjusts policy parameters until optimal routing strategy is achieved.

TABLE I. TE IN SDN USING RL – SUMMARY OF FINDINGS

TE in SDN	Agent Algorithm	Main Contribution	MDP	Limitations	Plane
[29]	Q-learning, Support Vector Machine	The authors proposed an enhanced traffic flow monitoring in SDN using Q-learning and Support Vector Supervised Machine Learning Algorithm	Yes	The statistics tracker should have factored in control link and data link capacity utilization of the SD Networks	Application
[30]	Q-learning, Deep Q-Network (DQN)	The authors addressed the TCAM capacity issue in OpenFlow switches by determining which forwarding rules remains in the flow table and those processed by the SDN controller	Yes	The $\epsilon - greedy$ policy should have given more value for exploration to balance the dynamics of the action taken.	Controller
[50]	Q-learning	The authors improved bandwidth utilization and reduced flow latencies – NRENs case study network	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Not stated
[52]	Q-learning	The authors addressed network congestion in SDN by reselecting flow paths and changing flow table using predefined threshold	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Controller

[53]	Q-learning	The authors introduced fairness function in SDN for load-balancing in peak traffic conditions	Yes	One type of user that should not be ignored is a compromised user with network intrusions.	Not stated
[54]	Q-learning, Bayesian Network	The authors used the Bayesian network to predict the degree of congestion and Q-learning for optimal action decision in SDN load-balancing framework	Yes	The rate of packet-in messages from the switches is enough parameter to predict the load congestion to the controller. Using the Bayesian Network will impede the idea of Reinforcement Learning	Controller
[56]	Q-learning	The authors proposed a dynamic switch migration algorithm with Q-learning in scaling the load on SDN controllers	Yes	No reward graph per episode to define the training and validation accuracy of the agent.	Not stated
[57]	Q-learning	The authors used an integrated DNN in Q-learning for load-balancing in SDN through flow forecasting	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Controller
[59]	Q-learning	The authors maximized table hit rates in a UDP flow entry eviction strategy in SDN by dynamically resizing sampling periods of critical parameters.	Yes	The scope of the state space definition is limited. Aside the size of the sampling period, the state of flows in the network will be an added metrics since UDP operates at the transport layer.	Not stated
[62]	SARSA, Bayesian Network	The authors proposed a switch migration prediction method based on Bayesian network and used with SARSA algorithm for overload-lighter load controller migration.	Yes	Comparing the modified SARSA algorithm to Q-learning in the research will have given a more comparative insight into the results of the research.	Not stated
[64]	SARSA	The authors proposed a QoS-aware adaptive routing scheme using SARSA to provide fast convergence in QoS provisioning in SDN	Yes	The reward function of the MDP is not well defined. Secondary, comparing the results with other known algorithms will have given more credence to the α, γ values	Not stated
[65]	SARSA	The authors proposed a resource allocation technique in massive IoT through cognitive communication in SDN-enabled environment	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Not stated
[66]	SARSA	The authors proposed a network hop count technique in SDN to improve VS-routing through $\varepsilon - Greedy$ function	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Not stated
[68]	DQN	The authors proposed a flexible network management through dynamic controller placement technique in SDN	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Not stated
[69]	DQN	The authors used a DQN based switch and controller selection scheme for switch migration in distributed SDN controllers	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Not stated
[70]	DQN	The authors used DQN to find energy-efficient routing paths and load-balancing between SDN controllers	Yes	Though energy-saving and load balancing are metrices defined in this research, the extent of a controller's ability to balance the load can be added to the reward functionality.	Controller
[71]	DQN	The authors used two DQN agents to detect mice and elephant flows in an SDN-enabled data center	Yes	A comparative analysis using packet-in and packet-out messages in defining the state-action-reward pair will have added higher scope to the research.	Application
[74]	DQN	The authors proposed a DQN agent that predicts optimal traffic paths and future traffic demands using LSTM neural networks.	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Not stated
[31]	DDPG	The authors proposed an SDN architecture to maximize QoE using DDPG agent to enforce bandwidth adaption and path chosen for all multimedia flows	Yes	There is little mathematical modelling of the DDPG algorithm used in this research. The pseudocode is not stated mathematically for this research. The parameters for simulation set up was not well defined in this	Controller

				research	
[80]	DDPG	The authors adopted the DDPG agent for dynamic routing in feature extraction with FFNN in the actor-critic network of the agent.	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Not stated
[82]	DDPG	The authors proposed a DDPG-EREP algorithm with dynamic planning of the experience pool capacity using the current iteration number of the sampling size	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Not stated
[83]	DDPG	The authors proposed a DQSP using DDPG algorithm with added intelligent layer above the control layer for routing policy optimization in SDN	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Application
[86]	DDPG	The authors proposed an iTE which leverages on ICN to optimize traffic distribution in SDN through the PDM module in the controller	Yes	The action space definition should have included the flow path selection procedure aside the split ratio for the i-th destination node.	Controller
[88]	DDPG	The authors used a DDPG agent to receive a deadline-aware data transfers from SDN switches and schedules subsequent flows by initiating a pacing rate at the source of the flows	Yes	This research can be extended to multi-path routing using AOMDV protocol	No stated
[89]	DQN, DDPG	The authors proposed a DRL-R based on DDPG-DQN agent to allocate cache and bandwidth in the SDN to improve routing performance	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	Controller
[98]	MADDPG	The authors proposed TCCA-MADDPG algorithm to reduce the channel interference between links by considering the policies of neighbouring agents using multi-agent strategy	Yes	The TCCA-MADDPG should have been compared with DDPG and not DQN since both TCCA-MADDPG and DDPG work in continuous environment.	Not stated
[99]	MADDPG	The authors proposed a MADDPG for effective traffic load engineering in SDN-IoV using a real-time distributed cooperative assignment approach via the Actor-Critic network	No	Since MDP was not used to mathematically define the network parameters, the measuring metrics for success is not well defined.	No stated

IV. Open Research Issues

In this section, we looked at the research gaps identified after the review. From the review summary shown in Table I, it is conclusive that, SDN-based TE solutions using RL agents has the potential to eliminate completely network degradation and provide a network recommender system for end users. From this review, some future research issues exist.

A. RL Agent Implementation

From the review RL agents are designed and situated at the control or application plane of the SDN architecture. For a more efficient and pro-active TE solutions, new SDN design architectures can situate the RL agent as mini-embedded applications adapted to dedicated forwarding devices with oversight from the SDN controller. With performance comparison based on end-to-end delay and response time [47], data plane based RL agents will enable a faster network congestion detection and prevention since the agents are closer to the forwarding devices.

B. RL Agent Algorithm

For TE, most RL agents use model-free based algorithms for policy enforcement and rewards. Though model-based algorithms have high computational complexities, a hybrid architecture that enables the RL agent to select either algorithm based on reward has a research value. Using trial-or-error and referencing a model will give more intelligence to the RL agent. The agent will have the capacity to decide the algorithm to activate based on network complexity and the priority of applications.

C. Multi-Agent Reinforcement Learning

For faster convergence and collaborative learning, MARL solutions in TE though complex is the future in solving network related routing and load-balancing in SDN architecture. The advent of connected devices will only increase with time. MARL agents from review have limited research [98][99] TE solutions in SDN. MARL when proposed efficiently can segment the network into smaller units with multi-agent capabilities.

V. Conclusion

Software-Defined Networks (SDN) has emerged to give more control in network management by separating the control layer from the forwarding devices. This separation has given a centralized programmable supervisory role to the controller and a flexible management of network flows in forwarding devices. In regulating the behaviour of data transmitted over the network, we discussed the relevance of Reinforcement Learning in SDN for Traffic Engineering. This paper explained major reviews using RL techniques in network traffic management and the action of agents on the environment for rewards and new states. The review further detailed the mathematical modelling of agents and environment using the Markov Decision Process (MDP). We illustrated with diagrams SARL and MARL agents and detailed their importance in regards to TE.

With Reinforcement Learning, agents are modelled in a controlled loop to take sequence of actions on the environment to receive future rewards and a new state. The agent must exploit and explore the stochastic environment through determined actions that will lead to a faster convergence. From the review, the paper offers future research options for optimal Traffic Engineering solutions in SDN.

REFERENCES

[1] Zanella, Andrea, et al. "Internet of things for smart cities." *IEEE Internet of Things journal* 1.1 (2014): 22-32

[2] Salem, Mohammed A., et al. "M2M in 5G Communication Networks: Characteristics, Applications, Taxonomy, Technologies, and Future Challenges." *Fundamental and Supportive Technologies for 5G Mobile Networks*. IGI Global, 2020. 309-321.

[3] Mattisson, Sven. "Overview of 5G requirements and future wireless networks." *ESSCIRC 2017-43rd IEEE European Solid State Circuits Conference*. IEEE, 2017.

[4] Ji, Hyoungju, et al. "Ultra-reliable and low-latency communications in 5G downlink: Physical layer aspects." *IEEE Wireless Communications* 25.3 (2018): 124-130.

[5] Busari, Sherif Adeshina, et al. "5G millimeter-wave mobile broadband: Performance and challenges." *IEEE Communications Magazine* 56.6 (2018): 137-143.

[6] Jungnickel, Volker, et al. "The role of small cells, coordinated multipoint, and massive MIMO in 5G." *IEEE communications magazine* 52.5 (2014): 44-51.

[7] Alencar, Felipe, et al. "How Software Aging affects SDN: A view on the controllers." *2014 Global Information Infrastructure and Networking Symposium (GIIS)*. IEEE, 2014.

[8] Kim, Hyojoon, and Nick Feamster. "Improving network management with software defined networking." *IEEE Communications Magazine* 51.2 (2013): 114-119.

[9] Yeganeh, Soheil Hassas, Amin Tootoonchian, and Yashar Ganjali. "On scalability of software-defined networking." *IEEE Communications Magazine* 51.2 (2013): 136-141.

[10] Kim, Hyojoon, and Nick Feamster. "Improving network management with software defined networking." *IEEE Communications Magazine* 51.2 (2013): 114-119

[11] Pruss, R. M., Mcdowall, J. E., Medved, J., & Abrahams, L. (2015). *U.S. Patent No. 9,047,143*. Washington, DC: U.S. Patent and Trademark Office.

[12] Karakus, Murat, and Arjan Durresi. "Quality of service (QoS) in software defined networking (SDN): A survey." *Journal of Network and Computer Applications* 80 (2017): 200-218.

[13] Kassler, Andreas, et al. "Towards QoE-driven multimedia service negotiation and path optimization with software defined networking." *SoftCOM 2012, 20th International Conference on Software, Telecommunications and Computer Networks*. IEEE, 2012.

[14] Mahboob, Tahira, Young Rok Jung, and Min Young Chung. "Optimized Routing in Software Defined Networks–A Reinforcement Learning Approach." *International Conference on Ubiquitous Information Management and Communication*. Springer, Cham, 2019.

[15] Perera, Menuka, Kandaraj Piamrat, and Salima Hamma. "Network Traffic Classification using Machine Learning for Software Defined Networks." *Journées non thématiques GDR-RSD 2020*. 2020.

[16] Bernaille, Laurent, et al. "Traffic classification on the fly." *ACM SIGCOMM Computer Communication Review* 36.2 (2006): 23-26.

[17] Finsterbusch, Michael, et al. "A survey of payload-based traffic classification approaches." *IEEE Communications Surveys & Tutorials* 16.2 (2013): 1135-1156.

[18] Khondoker, Rahamatullah, et al. "Feature-based comparison and selection of Software Defined Networking (SDN) controllers." *2014 world congress on computer applications and information systems (WCCAIS)*. IEEE, 2014.

[19] Dey, Samrat Kumar, and Md Mahbubur Rahman. "Flow based anomaly detection in software defined networking: A deep learning approach with feature selection method." *2018 4th International Conference on Electrical Engineering and Information & Communication Technology (iCEEiCT)*. IEEE, 2018.

[20] Sutton, Richard S., and Andrew G. Barto. "Introduction to reinforcement learning. Vol. 135." *MIT press Cambridge* 5 (1998): 21-22.

[21] Szepesvári, Csaba. "Algorithms for reinforcement learning." *Synthesis lectures on artificial intelligence and machine learning* 4.1 (2010): 1-103.

[22] Stampa, G., Arias, M., Sánchez-Charles, D., Muntés-Mulero, V., & Cabellos, A. (2017). A deep-reinforcement learning approach for software-defined networking routing optimization. *arXiv preprint arXiv:1709.07080*.

[23] Møller, Martin Fodslette. "A scaled conjugate gradient algorithm for fast supervised learning." *Neural networks* 6.4 (1993): 525-533.

[24] Torrey, Lisa, and Matthew Taylor. "Teaching on a budget: Agents advising agents in reinforcement learning." *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. 2013.

[25] Puterman, Martin L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[26] Boutilier, Craig, Thomas Dean, and Steve Hanks. "Decision-theoretic planning: Structural assumptions and computational leverage." *Journal of Artificial Intelligence Research* 11 (1999): 1-94.

[27] Van Otterlo, Martijn, and Marco Wiering. "Reinforcement learning and markov decision processes." *Reinforcement learning*. Springer, Berlin, Heidelberg, 2012. 3-42.

[28] Zhang, Junjie, et al. "CFR-RL: Traffic engineering with reinforcement learning in SDN." *IEEE Journal on Selected Areas in Communications* 38.10 (2020): 2249-2259.

[29] Phan, Trung V., et al. "Q-DATA: Enhanced Traffic Flow Monitoring in Software-Defined Networks applying Q-learning." *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019.

[30] Mu, Ting-Yu, et al. "SDN flow entry management using reinforcement learning." *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 13.2 (2018): 1-23.

[31] Huang, Xiaohong, et al. "Deep reinforcement learning for multimedia traffic control in software defined networking." *IEEE Network* 32.6 (2018): 35-41.

[32] Khan, Asiya, Lingfen Sun, and Emmanuel Ifeachor. "QoE prediction model and its application in video quality adaptation over UMTS networks." *IEEE Transactions on Multimedia* 14.2 (2011): 431-442.

[33] Zhang, Hongming, and Tianyang Yu. "Taxonomy of Reinforcement Learning Algorithms." *Deep Reinforcement Learning*. Springer, Singapore, 2020. 125-133.

[34] Buşoniu, Lucian, et al. "Approximate reinforcement learning: An overview." *2011 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE, 2011.

[35] Polydoros, Athanasios S., and Lazaros Nalpantidis. "Survey of model-based reinforcement learning: Applications on robotics." *Journal of Intelligent & Robotic Systems* 86.2 (2017): 153-173.

[36] Clavera, Ignasi, et al. "Model-based reinforcement learning via meta-policy optimization." *Conference on Robot Learning*. PMLR, 2018.

[37] Degris, Thomas, Patrick M. Pilarski, and Richard S. Sutton. "Model-free reinforcement learning with continuous action in practice." *2012 American Control Conference (ACC)*. IEEE, 2012.

[38] Song, Zhao, and Wen Sun. "Efficient model-free reinforcement learning in metric spaces." *arXiv preprint arXiv:1905.00475* (2019).

[39] Akrour, Riad, et al. "Model-free trajectory optimization for reinforcement learning." *International Conference on Machine Learning*. PMLR, 2016.

[40] Sutton, Richard S., and Andrew G. Barto. "Introduction to reinforcement learning. Vol. 135." *MIT press Cambridge* 5 (1998): 21-22.

[41] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).

[42] Kumar, Arun, Navneet Paul, and S. N. Omkar. "Bipedal walking robot using deep deterministic policy gradient." *arXiv preprint arXiv:1807.05924* (2018).

[43] Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." *Machine learning* 8.3-4 (1992): 229-256.

[44] Chiesa, Marco, Guy Kindler, and Michael Schapira. "Traffic engineering with equal-cost-multipath: An algorithmic perspective." *IEEE/ACM Transactions on Networking* 25.2 (2016): 779-792.

[45] Salisbury, B. "TCAMs and OpenFlow-what every SDN practitioner must know." *See http://tinyurl. com/kjy99uw* (2012).

[46] Khan, Asiya, Lingfen Sun, and Emmanuel Ifeachor. "QoE prediction model and its application in video quality adaptation over UMTS networks." *IEEE Transactions on Multimedia* 14.2 (2011): 431-442.

[47] Chin, Tommy, Mohamed Rahouti, and Kaiqi Xiong. "Applying software-defined networking to minimize the end-to-end delay of network services." *ACM SIGAPP Applied Computing Review* 18.1 (2018): 30-40.

[48] Nachum, Ofir, et al. "Bridging the gap between value and policy based reinforcement learning." *arXiv preprint arXiv:1702.08892* (2017).

[49] Suthaharan, Shan. "Machine learning models and algorithms for big data classification." *Integr. Ser. Inf. Syst* 36 (2016): 1-12.

[50] Chavula, Josiah, Melissa Densmore, and Hussein Suleman. "Using SDN and reinforcement learning for traffic engineering in UbuntuNet Alliance." *2016 International Conference on Advances in Computing and Communication Engineering (ICACCE)*. IEEE, 2016.

[51] Barbehenn, Michael. "A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices." *IEEE transactions on computers* 47.2 (1998): 263.

[52] Kim, Seonhyeok, et al. "Congestion prevention mechanism based on Q-leaning for efficient routing in SDN." *2016 International Conference on Information Networking (ICOIN)*. IEEE, 2016.

[53] Tennakoon, Deepal, Suneth Karunarathna, and Brian Udugama. "Q-learning approach for load-balancing in software defined networks." *2018 Moratuwa engineering research conference (MERCon)*. IEEE, 2018.

[54] LIANG, Siyuan, et al. "Load Balancing Algorithm of Controller Based on SDN Architecture Under Machine Learning." *Journal of Systems Science and Information* 8.6 (2020): 578-588.

[55] Wang, Ke, Wai-Choong Wong, and Teck Yoong Chai. "A MANET routing protocol using Q-learning method integrated with Bayesian network." *2012 IEEE International Conference on Communication Systems (ICCS)*. IEEE, 2012.

[56] Min, Zhu, Qu Hua, and Zhao Jihong. "Dynamic switch migration algorithm with Q-learning towards scalable SDN control plane." *2017 9th International Conference on Wireless Communications and Signal Processing (WCSP)*. IEEE, 2017.

[57] Yu, Chen, et al. "Intelligent optimizing scheme for load balancing in software defined networks." *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*. IEEE, 2017.

[58] Smith, Brian L., and Michael J. Demetsky. "Traffic flow forecasting: comparison of modeling approaches." *Journal of transportation engineering* 123.4 (1997): 261-266.

[59] Choi, Hanhimnara, et al. "UDP Flow Entry Eviction Strategy Using Q-Learning in Software Defined Networking." *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020.

[60] Nadeau, Thomas D., and Ken Gray. *SDN: Software Defined Networks: an authoritative review of network programmability technologies*. " O'Reilly Media, Inc.", 2013.

[61] Hausknecht, Matthew, and Peter Stone. "Deep reinforcement learning in parameterized action space." *arXiv preprint arXiv:1511.04143* (2015).

[62] Yang, Shike, Haobin Shi, and Hengsheng Zhang. "Dynamic Load Balancing of Multiple Controller based on Intelligent Collaboration in SDN." *2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)*. IEEE, 2020.

[63] Li, Zhihua, et al. "Bayesian network-based virtual machines consolidation method." *Future Generation Computer Systems* 69 (2017): 75-87.

[64] Lin, Shih-Chun, et al. "QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach." *2016 IEEE International Conference on Services Computing (SCC)*. IEEE, 2016.

[65] Arruda, Carlos E., et al. "Enhanced Pub/Sub Communications for Massive IoT Traffic with SARSA Reinforcement Learning." *arXiv preprint arXiv:2101.00687* (2021).

[66] Yuan, Zhengwu, et al. "Research on Routing Optimization of SDN Network Using Reinforcement Learning Method." *2019 2nd International Conference on Safety Produce Informatization (IICSPI)*. IEEE, 2019.

[67] Saraph, Girish P., and Pushpraj Singh. "Traffic engineering using new VS routing scheme." *2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577)*. Vol. 2. IEEE, 2004.

[68] Wu, Yiwen, et al. "Deep Reinforcement Learning for Controller Placement in Software Defined Network." *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020.

[69] Yeo, Sangho, et al. "Achieving Balanced Load Distribution with Reinforcement Learning-Based Switch Migration in Distributed SDN Controllers." *Electronics* 10.2 (2021): 162.

[70] Yao, Zan, Ying Wang, and Xuesong Qiu. "DQN-based energy-efficient routing algorithm in software-defined data centers." *International Journal of Distributed Sensor Networks* 16.6 (2020): 1550147720935775.

[71] Fu, Qiongxiao, et al. "Deep Q-learning for routing schemes in SDN-based data center networks." *IEEE Access* 8 (2020): 103491-103499.

[72] Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network." *2017 International Conference on Engineering and Technology (ICET)*. Ieee, 2017.

[73] O'Shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." *arXiv preprint arXiv:1511.08458* (2015).

[74] Bouzidi, El Hocine, Abdelkader Outtagarts, and Rami Langar. "Deep reinforcement learning application for network latency management in software defined networks." *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019.

[75] Sundermeyer, Martin, Ralf Schlüter, and Hermann Ney. "LSTM neural networks for language modeling." *Thirteenth annual conference of the international speech communication association*. 2012.

[76] Lipton, Zachary C., et al. "Learning to diagnose with LSTM recurrent neural networks." *arXiv preprint arXiv:1511.03677* (2015).

[77] Pitaksanonkul, Anucha, et al. "DTR: A defect-tolerant routing algorithm." *Proceedings of the 26th ACM/IEEE Design Automation Conference*. 1989.

[78] Streijl, Robert C., Stefan Winkler, and David S. Hands. "Mean opinion score (MOS) revisited: methods and applications, limitations and alternatives." *Multimedia Systems* 22.2 (2016): 213-227.

[79] Silver, David, et al. "Deterministic policy gradient algorithms." *International conference on machine learning*. PMLR, 2014.

[80] Stampa, Giorgio, et al. "A deep-reinforcement learning approach for software-defined networking routing optimization." *arXiv preprint arXiv:1709.07080* (2017).

[81] Schmidt, Wouter F., Martin A. Kraaijveld, and Robert PW Duin. "Feed forward neural networks with random weights." *International Conference on Pattern Recognition*. IEEE COMPUTER SOCIETY PRESS, 1992.

[82] Lu, Xiaoye, et al. "SDN routing optimization based on improved Reinforcement learning." *Proceedings of the 2020 International Conference on Cyberspace Innovation of Advanced Technologies*. 2020.

[83] Guo, Xuancheng, et al. "Deep-reinforcement-learning-based QoS-aware secure routing for SDN-IoT." *IEEE Internet of Things Journal* 7.7 (2019): 6242-6251.

[84] Dhawan, Mohan, et al. "SPHINX: detecting security attacks in software-defined networks." *Ndss*. Vol. 15. 2015.

[85] Ashraf, Javed, and Seemab Latif. "Handling intrusion and DDoS attacks in Software Defined Networks using machine learning techniques." *2014 National Software Engineering Conference*. IEEE, 2014.

[86] Zhang, Qingyi, et al. "Intelligent Content-Aware Traffic Engineering for SDN: An AI-Driven Approach." *IEEE Network* 34.3 (2020): 186-193.

[87] Dannewitz, Christian, et al. "Network of information (netinf)–an information-centric networking architecture." *Computer Communications* 36.7 (2013): 721-735.

[88] Ghosal, Gaurav R., et al. "A Deep Deterministic Policy Gradient Based Network Scheduler For Deadline-Driven Data Transfers." *2020 IFIP Networking Conference (Networking)*. IEEE, 2020.

[89] Xu, Chunlei, Weijin Zhuang, and Hong Zhang. "A Deep-reinforcement Learning Approach for SDN Routing Optimization." *Proceedings of the 4th International Conference on Computer Science and Application Engineering*. 2020.

[90] Buşoniu, Lucian, Robert Babuška, and Bart De Schutter. "Multi-agent reinforcement learning: An overview." *Innovations in multi-agent systems and applications-1* (2010): 183-221.

[91] Christianos, Filippos, et al. "Scaling Multi-Agent Reinforcement Learning with Selective Parameter Sharing." *arXiv preprint arXiv:2102.07475* (2021).

[92] Omidshafiei, Shayegan, et al. "Learning to teach in cooperative multiagent reinforcement learning." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. No. 01. 2019.

[93] Busoniu, Lucian, Robert Babuska, and Bart De Schutter. "A comprehensive survey of multiagent reinforcement learning." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008): 156-172.

[94] Gupta, Jayesh K., Maxim Egorov, and Mykel Kochenderfer. "Cooperative multi-agent control using deep reinforcement learning." *International Conference on Autonomous Agents and Multiagent Systems*. Springer, Cham, 2017.

[95] Egorov, Maxim. "Multi-agent deep reinforcement learning." *CS231n: convolutional neural networks for visual recognition* (2016): 1-8.

[96] Chu, Tianshu, et al. "Multi-agent deep reinforcement learning for large-scale traffic signal control." *IEEE Transactions on Intelligent Transportation Systems* 21.3 (2019): 1086-1095.

[97] Lowe, Ryan, et al. "Multi-agent actor-critic for mixed cooperative-competitive environments." *arXiv preprint arXiv:1706.02275* (2017)

[98] Wu, Tong, et al. "Joint Traffic Control and Multi-Channel Reassignment for Core Backbone Network in SDN-IoT: A Multi-Agent Deep Reinforcement Learning Approach." *IEEE Transactions on Network Science and Engineering* (2020).

[99] Yuan, Tingting, et al. "Dynamic Controller Assignment in Software Defined Internet of Vehicles through Multi-Agent Deep Reinforcement Learning." *IEEE Transactions on Network and Service Management* (2020).

[100] Jiacheng, Chen, et al. "Software defined Internet of vehicles: Architecture, challenges and solutions." (2016): 14-26.

[101] Dey, Ayon. "Machine learning algorithms: a review." *International Journal of Computer Science and Information Technologies* 7.3 (2016): 1174-1179.

[102] Van Engelen, Jesper E., and Holger H. Hoos. "A survey on semi-supervised learning." *Machine Learning* 109.2 (2020): 373-440.

[103] Sen, Pratap Chandra, Mahimarnab Hajra, and Mitadru Ghosh. "Supervised classification algorithms in machine learning: A survey and review." *Emerging technology in modelling and graphics*. Springer, Singapore, 2020. 99-111.

[104] Khanum, Memoona, et al. "A survey on unsupervised machine learning algorithms for automation, classification and maintenance." *International Journal of Computer Applications* 119.13 (2015).

[105] Bellemare, Marc G., Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning." *International Conference on Machine Learning*. PMLR, 2017.

[106] Wang, Haoran, Thaleia Zariphopoulou, and Xun Yu Zhou. "Exploration versus exploitation in reinforcement learning: a stochastic control approach." *Available at SSRN 3316387* (2019).

[107] Colas, Cédric, Olivier Sigaud, and Pierre-Yves Oudeyer. "Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms." *International Conference on Machine Learning*. PMLR, 2018.

[108] Holcomb, Sean D., et al. "Overview on deepmind and its alphago zero ai." *Proceedings of the 2018 international conference on big data and education*. 2018.