Q-placement: Reinforcement-Learning-Based Service Placement in Software-Defined Networks

Ziyao Zhang*, Liang Ma†, Kin K. Leung*, Leandros Tassiulas‡, and Jeremy Tucker §
*Imperial College London, London, United Kingdom. Email: {ziyao.zhang15, kin.leung}@imperial.ac.uk
†IBM T. J. Watson Research Center, Yorktown Heights, NY, United States. Email: maliang@us.ibm.com
‡Yale University, New Haven, CT, United States. Email: leandros.tassiulas@yale.edu
§UK Defence Science and Technology Laboratory, Salisbury, United Kingdom. Email: jtucker@mail.dstl.gov.uk

***Abstract*—In software-defined networking (SDN) paradigm, where the control and data plane are separated, the scalability of the SDN controller in the control plane is critical and can affect the overall network performance significantly. To improve controller scalability, efforts have been put into enhancing the capability of SDN switches in the data plane, to make them more autonomous in providing routine services without consulting the controller. In this regard, we investigate the service placement problem on SDN switches aiming at minimizing the average accumulated service costs for end users. To solve this problem, we propose a novel reinforcement-learning-based algorithm with guaranteed performance and convergence rate, called Q-*placement*. Comparing to traditional optimization techniques, Q-placement exhibits many appealing features, such as performance-tuneable optimization and off-the-shelf implementation. Extensive evaluations show that Q-placement consistently outperforms benchmarks and other state-of-the-art algorithms in both synthetic and real networks. Moreover, these evaluations reveal insights into how the network topological properties (e.g., density), servicing capacities, and controller's roles affect the accumulated service costs, which is useful in service planning tasks.**

I. Introduction

Software-Defined Networking (SDN) [1] [2] [3] [4], an emerging networking architecture, significantly improves the network performance due to its programmable network management, easy reconfiguration, and on-demand resource allocation, which has therefore attracted considerable research interests. One key attribute that differentiates SDN from classic networks is the separation of the SDN's data and control plane. Specifically, in SDN, all control functionalities are implemented and abstracted in the *SDN controller*, which sits in the control plane, for operational decision making, while the data plane, consisting of *SDN switches*, only passively executes the instructions received from the control plane. Since the centralized SDN controller has full knowledge of the network status, it is able to make the global optimal decision. Yet, such centralized control suffers from major scalability issues. In particular, as a network grows, the number of service requests and operational constraints are likely to increase exponentially. Such high computation and communication requirements may impose substantial burden on the SDN controller, thus potentially resulting in significant performance degradation (e.g., delays) or even network failures.

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

To tackle this scalability issue, efforts have been put into reducing unnecessary SDN controller involvements in situations where switches are good enough for handling user requests. In this way, the controller can focus on important decision-making tasks that require high-level abstractions and centralized network-wide views. As such, SDN switches may take over most routine tasks and operations that require only local information. For example, Kandoo [5] is proposed to enable two types of applications to coexist, i.e., locally scoped applications and those require network-wide states. Moreover, there are also proposals [6] [7] suggesting that SDN switches should be equipped with general purpose CPUs to allow for greater flexibility and on-switch processing.

In this regard, we generalize different functionalities provided by SDN switches as *services*, each of which may associate with communication, computation and/or storage costs. We study the problem of placing these services in the network so as to minimize the accumulated service costs. Depending on the availability of resources of the SDN controller, it may decide, to what extent, it wishes to provide services to SDN switches, if such services are not installed on the switch. On the other hand, switches work in a cooperative way, i.e., a switch can find and use services installed on other switches.

To this end, we propose Q-*placement*, a reinforcement-learning-based algorithm with performance and convergence rate guarantees, to optimally decide where to place the services in an iterative manner. Comparing to traditional approaches, our method exhibits significant advantages. First, we do not have any specific assumptions or requirements on the network or the services considered. In contrast, traditional approaches generally impose restrictions, mostly on the network (e.g., network structures and size) or on services (e.g., service capacity and demand), so as to simplify the algorithm development task. Second, the decision-making process of Q-placement is *on-demand* and *pay-as-you-go*. Unlike traditional optimization algorithms with fixed optimization levels, Q-placement lets one decide the optimization level. This is achieved by tuning the number of iterations the algorithm runs. It is a desirable feature because the controller has the ability to decide exactly how much computation power it commits in order to achieve certain performance level. This flexibility is important especially in situations where the controller is overloaded. Third, the Q-placement algorithm is

fully compatible with the SDN architecture, i.e., the existing SDN infrastructure and protocol suite are *off-the-shelf* for its implementation. In fact, the Q-placement algorithm itself can be regarded as an application running on the SDN controller. All information needed by Q-placement are collected during routine network status updates between controller and switches through southbound interface (e.g., OpenFlow [8]). These appealing features make Q-placement suitable to our service placement problem in the SDN environment.

To validate the performance of Q-placement, we run a series of simulations extensively to compare the performance of Q-placement with other *state-of-the-art* algorithms. The network topologies used are built using both real and synthetic datasets. We also tune parameters related to network topologies and service patterns to explore different factors that may impact Q-placement's performance. The evaluation results demonstrate that Q-placement consistently performs the best among all algorithms tested in all scenarios.

The rest of the paper is organized as follows. We describe how the system works in Section II. Section III formally formulates the problem. Section IV describes the details of Q-placement. Section V presents the performance evaluation results. Finally, we review related work in Section VI and conclude the paper in Section VII.

II. System Description

Naturally, the SDN controller and the SDN switch are the most essential infrastructures in the service placement problem. Users are directly connected to the switches where they can submit requests for services. The switches are connected according to certain network topology, and switches are logically connected to the controller. The SDN controller acts as an information gatherer and decision maker, whereas the switches are servers that work cooperatively to serve the users and collect network status information. The controller decides and installs the services that the switches provide. The service placement decisions are made by the controller according to the given control objective. The responsibilities of the controller and the switches are described in details below.

A. SDN Controller

As an information gatherer, the controller has the global view of the network. The topology information is reflected as the stored routes and their associated costs in the routing information table. Furthermore, the controller knows the profiles of all switches in the network, such as service capacity and the connection speed of different interfaces. The controller also keeps track of the demands for services at each switch. This data is collected at switches and submitted to the controller during routine update process. As a decision maker, the controller makes use of the information collected and runs algorithms to make service placement decisions according to certain control objectives. The decision-making algorithms are essentially high-level applications offered by the SDN controller, which is programmable by the network administrator.

B. SDN Switch

The switches are responsible for assisting the controller in collecting relevant information. Second, each switch provides services to the users connecting to it. The term "service" generalizes a range of network functionalities that might be carried out by different physical network equipment in traditional network paradigm (e.g., the admission control function of the wireless access point or the routing function of routers). In order to clearly identify how a requested service is provided, the switches have to maintain an information base called service availability table, which indicates where a requested service can be found, either from current switch, from another switch, or from the SDN controller. The controller, on the other hand, installs service fetching rules on switches.

C. System

The system works as follows. First, the controller builds the network view and estimates the requirement for services based on the data collected by the switches. Then, the controller runs algorithms to determine the service placement arrangements and service fetching rules for switches. These service placement decisions remain valid until either the network view or the requirement for service changes. The controller learns about changes during routine update process with the switches, and replaces some of the services accordingly if needed.

III. Problem Formulation

A. Network Model

We formulate the SDN network in the data plane as an undirected graph of n vertices, where each vertex represents an SDN switch. These n vertices are connected via undirected links. The network topology is denoted by graph $\mathcal{G} = (V, E)$ (V/E: set of vertices/edges in $\mathcal{G}$, $|V| = n$). Without loss of generality, we assume that $\mathcal{G}$ is a connected graph without any isolated vertices. The SDN controller logically sits on top of $\mathcal{G}$ and can reach every node within the network.[1] Such network model is generic in that unlike [10] [11], we do not impose any structural restrictions on network topologies.

B. Service Costs and Link Weights

We associate weights to links to capture the corresponding access cost of services. Specifically, let $\mathcal{U} \in \mathbb{R}^{n \times n}$ be the symmetric adjacency matrix of the graph representing the switch network, where entry $u_{ij} \in \mathcal{U}$ denotes the service access weight between vertex v_i and vertex v_j. In addition, vector $\mathcal{C} \in \mathbb{R}^{n \times 1}$ contains the service access weights between the SDN controller and switches, with entry c_i denoting the service access weight between the controller and switch i.

Providing a service may consist of computation, storage and communication (access) costs. The controller leverages service access weights to reflect these costs. For example, if the controller deems that a switch is low in computation capacity, it may associate high service access weight to all links connected to this switch, in order to discourage other switches from submitting service requests to it. Therefore, service access weights are used to represent the *service costs*. In fact, this is a widely used technique in many real-world networks [12]. In

[1]The controller controls switches in in-band or out-of-band manners. In-band control means that control traffic is communicated between the controller and switches using existing data plane links, whereas out-of-band control means that the controller has dedicated direct control links to switches [9].

general, the service costs between the controller and switches are greater than the service costs between switches. This is to penalize unnecessary communication/computation overheads added to the controller, which could cause scalability issues. In practice, this implies that the controller, although with significantly more capabilities, does not wish to handle routine services that could be dealt with by switches. Thus, we call $\mathcal{C}$ the *penalty cost* vector. In this sense, the penalty cost vector reflects the level of controller's willingness to provide services. Based on the above, we argue that all types of costs are captured by $\mathcal{U}$ and $\mathcal{C}$, which simplifies the problem formulation.

C. Demand for Service

Apart from service costs, demand pattern for different services at different switches is another crucial factor to be considered when making service placement decisions. The demand for service data is gathered by switches, which count the number of requests received for different services during a time interval. These demand statistics are communicated to the controller via routine updates process. We use matrix $\Lambda \in \mathbb{R}^{n\times m}$ to denote the service demand at switches, where m is the total number of different services provided. Then, entry $\lambda_{ij} \in \Lambda$ represents the probability that switch i receives request for service j.

D. Value of Service (VoS)

The goal of the service placement problem is to provide services to all users in the SDN network with minimum costs given accessing penalty between the controller and switches (as specified by $\mathcal{C}$). Therefore, we need to have a measure of the *usefulness* of installed service that reflects this goal. Formally, the service placement of all switches is denoted by binary matrix $\mathcal{P} \in \mathbb{R}^{n\times m}$, where entry p_{ij} indicates whether service j is installed on switch i ($p_{ij} = 1$) or not ($p_{ij} = 0$). Let $L_{p_{ij}}$ be the set of switches that service j at node i serves (for $p_{ij} = 1$), which we call *beneficiary switches* of service j at node i. The set of beneficiary switches of a switch is identified by the controller using the cost information from $\mathcal{U}$ and $\mathcal{C}$. The *value of service (VoS)* of service j installed on switch i under the current service placement $\mathcal{P}$, denoted by $\vartheta_{L_{p_{ij}}}$, is defined as:

$$\vartheta_{L_{p_{ij}}} = \begin{cases} \sum_{k\in L_{p_{ij}}} (c_k - u_{ik})\lambda_{kj} & \text{if } p_{ij} = 1, \\ 0 & \text{if } p_{ij} = 0. \end{cases}$$

It can be seen that the VoS is the sum of demand-discounted difference of penalty costs and service costs of all beneficiary switches of switch i w.r.t. the installed service j. Then, we define the *accumulated VoS (aVoS)*, which is the sum of VoS of all services installed on switches in the network, as aVoS:= $\sum_{i=1}^{n}\sum_{j=1}^{m} \vartheta_{L_{p_{ij}}}$. For one service, the definition of VoS gives higher rewards to scenarios where the penalty cost is high. This is because the service cooperation among switches is encouraged when the controller sets high penalty costs to free itself.

E. Problem Formulation

To achieve the maximum aVoS, we formulate the service placement problem as a Markov Decision Process (MDP), for which we propose (in Section IV) an algorithm with performance guarantees. In particular, our MDP is characterized by a 4-tuple $(\mathcal{S}, \mathcal{A}, P, R)$, detailed as below:

- $\mathcal{S}$ is the finite state space. In our problem, a state corresponds to the decision of the removal of an installed service. Specifically, $s_{ij} \in \mathcal{S}$ represents the state where a decision that service j at switch i is deleted. Recall that m is the total number of services considered. Then, there are in total $n \times m$ possible states in the service placement problem.
- $\mathcal{A}$ is the finite action space. An action w.r.t. a state is defined as installing a new service which is not available at the current switch, to replace the removed service. Specifically, action $a_{ij}^{(w)} \in \mathcal{A}$ means installing service w at switch i when service j is removed. The number of actions at each state is $m-(\kappa_i-1)$, where κ_i is the maximum number of services that can be installed on switch i.
- P is the state transition strategy, which is designed and called the *Greedy-Shifter* in our formulation. The main idea behind Greedy-Shifter is that the next state after an action is primarily decided by VoSes of currently installed services. Specifically, the state at the next time instant (the decision of removing an installed service at a switch) is decided, either by choosing the installed service that has the least VoS, with probability $1 - \delta$; or by choosing an installed service randomly, with probability δ. The value of δ is close to 0.
- R represents the immediate reward function associated with state-action pairs, which we define as $R(s, a)$, where $s \in \mathcal{S}$ and $a \in \mathcal{A}$. $R(s, a)$ is calculated as the difference in VoS before and after the service replacement takes place. In particular, when service j in switch i is replaced with service w, the immediate reward function for this state-action pair, denoted as $R(s_{ij}, a_{ij}^{(w)})$, is $R(s_{ij}, a_{ij}^{(w)}) = \vartheta_{L_{p_{iw}}} - \vartheta_{L_{p_{ij}}}$.

At the start of the MDP, all switches are installed with services to their **full capacity**. However, the initial placement is random, thus unoptimized. The MDP enters a different state after a state-action pair. Each state transition generates a positive or negative reward. A positive/negative reward causes an increase/decrease in aVoS. Eventually, we arrive at some states of the MDP where the immediate reward function R constantly fails to generate any positive rewards. The service placement problem then reaches the *State of Optimal Service Placement (SOSP)*, since no further replacement of services would cause an increase in aVoS. Therefore, the goal of the formulated MDP is to find a sequence of state-action pairs (policy) that eventually reaches SOSP, efficiently.

The *optimal action* at each state is defined as the action that gives the maximum long-term reward, which is the discounted sum of the expected immediate reward of all future state-action pairs from the current state. The reward for the state-action pair Δt steps ahead of the current state is discounted by $\gamma^{\Delta t}$, where γ is called the *discount factor* and $0 < \gamma < 1$. Here, γ trades off the importance between the current and the future reward. Therefore, starting from an initial state s_0, the problem is formulated to maximize the long-term accumulated reward expressed in the following Bellman equation by selecting a sequence of actions $\{a_t\}_{t=0}^{\infty}$:

$$V(s_0) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)|s_0\Big], \quad (1)$$

where s_t and a_t constitute the state-action pair at time t.

Discussions : For our problem, an alternative MDP formulation may define the state as the current service placements in the switch network. However, the corresponding total number of states can be as large as $\prod_{i=1}^{n} \binom{m}{\kappa_i}$, which grows exponentially in both n and m. In comparison, the number of states in our formulation is only $n \times m$, and thus grows linearly in n or m. Therefore, our service-removal-based state definition significantly reduces the size of state space, thus shortening the running time of the algorithm.

IV. Q-PLACEMENT

The *Q-placement* algorithm is based on the classic model-free reinforcement learning technique *Q-learning* [13], which is used to find the optimal state-action policy for any MDP. Q-learning is proved to be optimal under certain conditions. The idea of Q-learning is that an agent can move from state to state by taking certain actions. The agent attempts to maximize its accumulated reward by applying different state-action pairs and learn the optimal action at each state.

Q-placement uses the *Q-function* to estimate the quality of a state-action combination:

$$\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}.$$

In particular, the optimal Q-function for a state-action pair in Q-placement is defined as:

$$Q^*(s,a) = \mathbb{E}[R(s,a) + \gamma \max_{a' \in A_{s'}} Q^*(s', a')], \quad (2)$$

where s' and a' is a state-action pair at the next time instant, and $A_{s'}$ is the set of actions available at the next state s'. All learned values from the Q-function constitute the *Q-matrix*. Based on the Q-matrix, we know that the optimal value of (1) is $V^*(s_0) = \max_{a \in A_{s_0}} Q^*(s_0, a)$. At the start of the algorithm, the Q-matrix is initialized with all 0 entries. Therefore, the agent's Q-function, denoted as $Q(s,a)$, can be substantially different from the optimal Q-function at the beginning, due to the lack of experiences. However, the *value iteration update* of the Q-placement algorithm makes $Q(s,a) \rightarrow Q^*(s,a)$ when $t \rightarrow \infty$. In particular, as the learning process proceeds, the Q-matrix is updated using the following rule:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[R(s,a) + \gamma \max_{a' \in A_{s'}} Q(s', a')], \quad (3)$$

where α ($0 < \alpha < 1$) is the learning rate of Q-placement.

The procedures of the Q-placement algorithm is summarized in Algorithm 1. An important element in Algorithm 1 is the *exploration policy*. In simple terms, an exploration policy tells the agent how to choose an action given a state. At the beginning of Q-placement, the Q-matrix is sparse. Nevertheless, the agent's experiences build up over time, as manifested by the growing number of non-zero Q-values in Q-matrix. At each state, the agent decides the action either by choosing randomly from all available actions if there is no positive Q-value for any state-action pair, or by certain strategy that exploits past experiences when positive Q-value exists for some state-action pairs.

Specifically, for each state, Q-placement first identifies the set of actions with positive Q-value (see line 8). If there is no action with a positive Q-value, the agent chooses an action randomly (see line 10). Otherwise, the agent decides how to exploit those actions with positive Q-values using the *ϵ-greedy algorithm* [14], which makes sure that the agent makes sufficient explorations and offers rapid convergence rate (see line 12). With ϵ-greedy exploration policy, given a state, the agent chooses the action with the maximum Q-value with probability $1-\epsilon$, and chooses a random action with probability ϵ, where ϵ is usually very small. As the name suggests, this exploration policy is greedy in the sense that it tries, with high probability, to exploit actions with high known rewards to take advantage of the *learned experiences*.

As the Q-placement algorithm proceeds, all entries in Q-matrix eventually approach stable values. This happens when the algorithm converges. In practice, this suggests that the MDP enters SOSP, since no additional VoS would be brought to the network by further service replacement. In fact, we are able to tell how close the service placement is to the optimality and how fast this is achieved, as stated below.

Optimality: One advantage of using Q-learning for solving MDP is that there exists provable optimality guarantee. Specifically, it has been proved in [13] that the update rule (i.e., (3)) in Q-learning converges with probability 1 to the optimal Q-function w.r.t. the Bellman equation, as long as all state-actions pairs are visited infinitely often. In the Q-placement algorithm, the Greedy-Shifter policy ensures that all states would be visited infinitely as time elapses. On the other hand, ϵ-greedy exploration strategy is a justified exploration policy that meets this optimality requirement as well. Therefore, we have the assurance that Q-placement eventually converges and the Bellman equation is maximized.

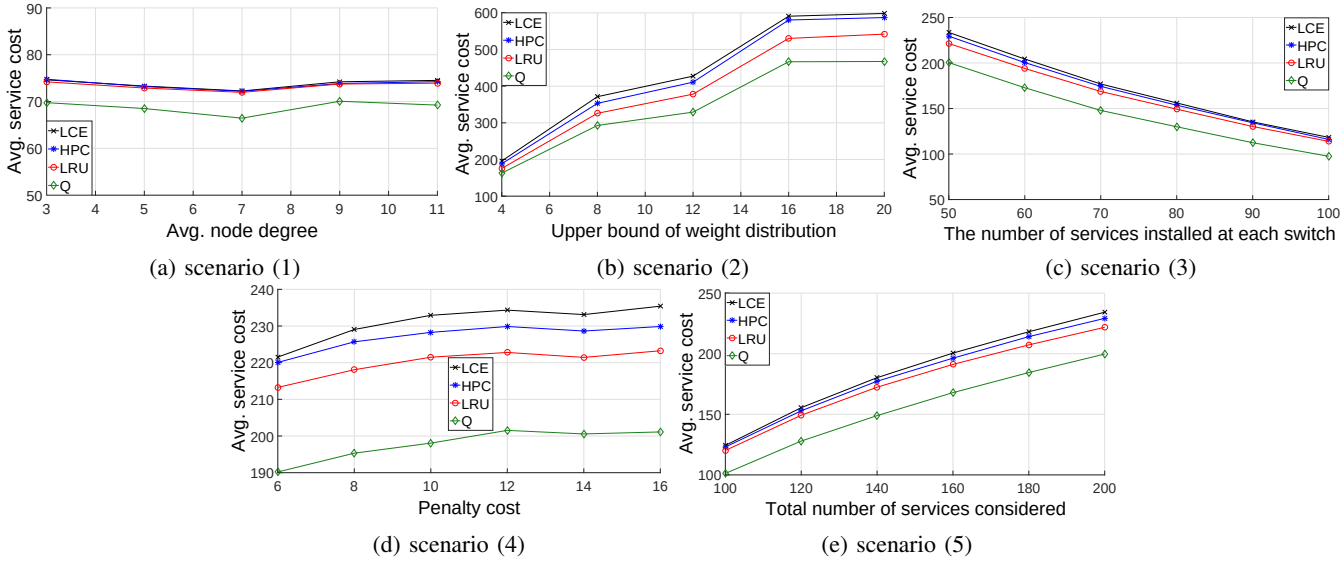
Rate of convergence: We now discuss the convergence rate of the Q-placement algorithm. The authors in [15] proved that only on the order of $(N \log(1/\xi)/\xi^2)(\log(N) + \log\log(1/\xi))$ iterations are sufficient for Q-learning to come with ξ of the optimal policy, where N is the number of states of the MDP. This guarantees that the Q-placement algorithm experiences a rather rapid convergence to optimality.

V. PERFORMANCE EVALUATIONS

A. Evaluation Settings

In this section, we present evaluation results of the performance of the Q-placement algorithm in various scenarios. In different evaluation scenarios, we keep most parameters to the default values but vary the following in the five scenarios considered: (1) average node degree of the network topology; (2) upper bound of link weight distribution; (3) service capacity of switches; (4) penalty cost vector; (5) the number of services considered. Detailed parameter settings for different scenarios are summarized in Table I.

We compare the performance of our Q-placement algorithm to some existing classic or state-of-the-art service/data placement algorithms. Specifically, we use *Least Recent Used (LRU)*, *Leave Copy Everywhere (LCE)*, and *Hop-based Probabilistic Caching (HPC)* algorithms as benchmarks. LRU is



(a) scenario (1) (b) scenario (2) (c) scenario (3)

(d) scenario (4) (e) scenario (5)

Fig. 1: Simulation results in different scenarios

Algorithm 1: Q-placement

input : network topology $\mathcal{G}$; initial service placement $\mathcal{P}_0$; penalty cost vector $\mathcal{C}$; service demand matrix $\mathcal{D}$.
output: optimized service placement $\mathcal{P}$.

Calculate VoS of all services given $\mathcal{P}_0$ (section III-D);
Determine the number of iterations T (time instants);
$t = 0$;
Initialize Q-matrix with all 0 entries;
while $t \leq T$ **do**
  foreach *time instant* t **do**
    Determine current state s according to Greedy-Shifter strategy;
    Construct set $\mathcal{A}_c = \{a : a \in \mathcal{A}, Q(s,a) > 0\}$;
    if $\mathcal{A}_c = \emptyset$ **then**
      Determine action $a \in \mathcal{A}$ by choosing a service w, which is not yet installed on the current switch, randomly, to replace the deleted service specified by s;
    else
      Determine action $a \in \mathcal{A}_c$ according to ϵ-greedy algorithm;
    end
    Update VoS of services affected by the replacement;
    Update $Q(s,a)$ according to (3);
  end
end

a classic cache replacement algorithm widely used in web-caching applications. LCE is implemented as the default cache coordination policy in many proposed Information Centric-Network (ICN) architectures. HPC is one of many state-of-the-art algorithms, which addresses data object placement problem with exceptional performance.

Both real and synthetic datasets are used to generate switch networks. For simulations involving varying network parameters, i.e., scenario (1) and (2), we use synthetic networks generated by the *Erdös-Rényi (ER)* network model. For scenarios (3), (4), and (5), we use real datasets collected by the Rocketfuel project [16].

We assume that every switch receives a request for service at every time instant, i.e., all switches receive requests at the same rate. The requests for different services at a switch are generated according to the demand for service distribution of that switch. The requested services are provided by any one of the following, whichever incurs the minimum cost: the switch receiving the request, other switches with the required service, or the SDN controller. The costs for providing services using the service placement arrangements by LRU, LCE, HPC, and Q-placement are recorded, respectively. This process is repeated for 1000 times and the average results are reported.

TABLE I: Evaluation Scenarios

Scenario	Varying parameter and their default value
(1)	avg. node degree ranges from 3 to 11 (default value: 4)
(2)	link weight distribution follows uniform distribution between 1 and the upper bound, with upper bound ranging from 4 to 20 (default value: 10)
(3)	the number of services installed at each switch ranges from 50 to 100 (default value: 50)
(4)	penalty cost ranges from 6 to 16 (default value: 10)
(5)	total number of services considered ranges from 100 to 200 (default value: 100)

Throughout our simulation, we choose 200 switches in the SDN network. The demand for service at switches follows the Zipf-Mandelbrot distribution with shape parameter μ and shift parameter q. The demand for r-th most popular service at a switch is proportional to $(q+r)^{\mu}$. For example, if service j is r-th most popular in switch i, the service demand for j, denoted by λ_{ij}, is calculated as $\lambda_{ij} = \frac{(q+r)^{\mu}}{\sum_{x=1}^{\kappa_i}(q+x)^{\mu}}$. We assume that $\mu = 0.85$ for all switches and q be a random integer between 10 and 20 for each switch.

Note that the network settings used for the simulation do not imply any specific requirements for the network or service-related attributes. A main advantage of Q-placement is its ability to deal with highly heterogeneous inputs.

B. Evaluation Results

Simulation results are presented in Fig.1(a) - Fig.1(e) under the five different scenarios.

1) Superior performance of Q-placement: Q-placement consistently performs better than other classic or state-of-the-art algorithms, and offers up to 20% performance improvements over the second best algorithm tested. Q-placement is particularly helpful in situations where the service cost is high, as demonstrated by the widening performance gap when the upper bound of weight distribution increases.

2) The invariance of service costs with varying network connectivity: Surprisingly, the average service cost does not appear to vary with increasing average node degree (scenario (1)). This is true for all tested algorithms. This revelation implies that although enhanced network connectivity makes services more widely available, it does not necessarily help reduce service costs. In contrast, improving service capacity of switches proves to be a better investment. This is confirmed by Fig.1(c), where the service capacity is doubled from 50 to 100, and the average service costs is halved.

3) Higher penalty cost does not always lead to higher average service cost: The penalty cost is determined by the network administrator to discourage switches from submitting unnecessary service requests. From Fig.1(d), we observe that there is no further increase in average service cost when the penalty cost exceeds 12. This suggests either of the following: (1) the majority of the services are provided by switches already, or (2) the number of requests sent to the controller decreases as the penalty cost increases. Both cases lead to the improved controller scalability. In addition, we only see a 5% increase in the average service cost when the penalty cost is doubled from 6 to 12. This suggests that the scalability of controller can be improved without triggering a huge increase in the average service cost.

VI. Related Work

Most works that are related to ours are in the area of cooperative caching. These problems are discussed in the context of web caching and Content-Centric Networks, where the caches store data objects, and they assist each other in delivering these contents to the users. The data objects can be compared to services in our problem.

Authors in [11] conducted an algorithmic analysis on the data placement problem which is similar to our problem. However, their proposed solution is a 10-approximation algorithm, i.e., the bound is too loose. The work in [17] [18] propose the idea of content router, which is similar to the concept of SDN switches. However, these works only focus on designing schemes without any performance guarantees. Similarly, [19] [20] proposed different methods, with different focuses, to implement content routers. They show different levels of improvements over existing algorithms, but fail to justify some particular assumptions, such as the $(k-1)$-hop cache cooperation range. These works also lack any forms of performance quantification, other than emulations. The contribution of [21] is that the authors discover the impact of some topology characteristics and heterogeneous capacities of the network on caching collaboration. However, the drawback is that these conclusions only apply to a specific hierarchical network topologies. The work in [22] discusses the caching collaboration network, also in the context of SDN. However, it lacks novelty in both problem formulation and solution, compare to Q-placement.

VII. Conclusions

We studied the service placement problem among SDN switches aiming at improving scalability of the SDN controller and minimizing the total service cost. We formulated the problem as an MDP problem with the reduced state space and proposed a reinforcement-learning-based algorithm with performance guarantees. Evaluations confirm the high efficiency of our Q-placement in reducing the average service cost comparing to other algorithms.

References

[1] N. McKeown. Software-defined networks and the maturing of the internet. [Online]. Available: https://tv.theiet.org/?videoid=5447

[2] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[3] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[4] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, 2007, pp. 1–12.

[5] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *ACM HotSDN*, 2012.

[6] G. Lu, R. Miao, Y. Xiong, and C. Guo, "Using CPU as a traffic coprocessing unit in commodity switches," in *ACM HotSDN*, 2012.

[7] J. C. Mogul and P. Congdon, "Hey, you darned counters!: get off my asic!" in *ACM HotSDN*, 2012, pp. 25–30.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[9] A. Soltani and C. F. Bazlamacci, "Hyfi: Hybrid flow initiation in software defined networks," in *IEEE ICICS*, 2014.

[10] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *IEEE INFOCOM*, 2010.

[11] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," *SIAM Journal on Computing*, vol. 38, no. 4, pp. 1411–1429, 2008.

[12] U. D. Black, *IP routing protocols: RIP, OSPF, BGP, PNNI, and Cisco routing protocols*. Prentice Hall Professional, 2000.

[13] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[14] S. B. Thrun, "Efficient exploration in reinforcement learning," Tech. Rep., 1992.

[15] M. J. Kearns and S. P. Singh, "Finite-sample convergence rates for Q-learning and indirect algorithms," in *Advances in Neural Information Processing Systems*, 1999, pp. 996–1002.

[16] "Rocketfuel: An ISP topology mapping engine," University of Washington, 2002. [Online]. Available: http://www.cs.washington.edu/research/networking/rocketfuel/interactive/

[17] W. Wong, M. Giraldi, M. F. Magalhaes, and J. Kangasharju, "Content routers: Fetching data on network path," in *IEEE ICC*, 2011.

[18] W. Wong, L. Wang, and J. Kangasharju, "Neighborhood search and admission control in cooperative caching networks," in *IEEE GLOBECOM*, 2012.

[19] Z. Ming, M. Xu, and D. Wang, "Age-based cooperative caching in information-centric networking," in *IEEE ICCCN*, 2014.

[20] Z. Li and G. Simon, "Time-shifted tv in content centric networks: The case for cooperative in-network caching," in *IEEE ICC*, 2011.

[21] J. Dai, Z. Hu, B. Li, J. Liu, and B. Li, "Collaborative hierarchical caching with dynamic request routing for massive content distribution," in *IEEE INFOCOM*, 2012.

[22] Y. Cui, J. Song, M. Li, Q. Ren, Y. Zhang, and X. Cai, "SDN-based big data caching in ISP networks," *IEEE Transactions on Big Data*, vol. PP, no. 99, pp. 1–1, 2017.