DYNAMIC SDN CONTROLLER PLACEMENT BASED ON DEEP REINFORCEMENT LEARNING

Fan Shen, Levi Perigo

Department of Computer Science, University of Colorado Boulder, CO 80309, USA

ABSTRACT

Software-defined Networking (SDN) is a revolutionary network architecture whose benefits stem partly from separating the data plane and control plane. In this scheme, the control functionalities are relocated to a logically centralized SDN controller which makes efficient and globally optimal forwarding decisions for network devices. Despite the fact that network virtualization technologies enable elastic capacity engineering and seamless fault recovery of the SDN controller, an optimal controller placement strategy that can adapt to changes in networks is an important but underexplored research topic. This paper proposes a novel deep reinforcement learning-based model that dynamically and strategically adjusts the location of the controller to minimize the OpenFlow latency in a virtualized environment. The experimental results demonstrate that the proposed strategy out performs both a random strategy and a generic strategy. Furthermore, this paper provides detailed instructions on how to implement the proposed model in real-world software-defined networks.

KEYWORDS

SDN, controller placement problem, latency, deep reinforcement learning, DQN

1. INTRODUCTION

In Software-defined Networking (SDN), the control plane of network devices is separated from the hardware and moved to a logically centralized and physically distributed location. The separation of the control plane and data plane has many advantages, including optimal decisions for the network because of the global network view from the SDN controller, more openness and programmability, and lower CapEx and OpEx. The way a network device in traditional SDN with OpenFlow deals with an incoming packet is as follows: if the packet matches a flow entry in its flow table, it takes the action that is specified in that flow entry; otherwise, it forwards the packet to the controller, and northbound applications instruct the controller how to handle the packet and a flow entry could also be installed on switches. As the control plane functionalities of network devices are moved to the logically centralized controller in SDN, it is crucial to guarantee the non-stop operation and sufficient processing capability of the controller. Network function virtualization (NFV) has been shown to be a viable solution for SDN controllers that require elastic infrastructure resources [1] [2] [3]. In addition, by using network virtualization technologies (virtual machines (VM), containers) and creating the SDN controller as a virtual network function (VNF), the failure of SDN controllers can be seamlessly mitigated, because, if the SDN controller is a container or VM hosted in a virtualized server environment, then failures can be reduced due to the redundant nature of the virtualized infrastructure. Although issues like capacity adjustment and fault tolerance can be mitigated by an SDN controller VNF to provide network resilience, the location of the SDN controller also needs to be strategically selected to achieve service agility by minimizing the latency of the traffic between switches and controller, which adds more granularity to the standard placement strategy which relies on servers.

The SDN controller placement problem is mainly about the optimization of the location and the number of controllers and the switch-controller mapping, in order to achieve the best network performance regarding the metrics of interests. Many studies attempt to solve the SDN controller placement problem in the network provisioning phase, thus analytically giving an optimal but static placement solution in terms of their objective function with some constraints. However, the operational network dynamically changes, for example, the available bandwidth of paths in the physical network between switches and SDN controllers are affected by dynamic patterns of traffic using those paths. Therefore, a static placement strategy is not optimal in the long term. Some existing works study dynamic SDN controller placement to adapt to network changes, but there are some open issues that need to be addressed. First, the dynamic placement decision is made based on recent but outdated network state information, since the future network state after taking the placement action is possibly different than the network state used for calculating the optimal placement solution, such models could generate sub-optimal placement strategy. Second, only switch-controller traffic is assumed to exist in the network between switches and SDN controllers, which is not completely true when considering an SDN controller VNF is used. Third, existing studies using deep reinforcement learning lack sufficient understanding about the mathematical model, and detailed implementation instructions for interactions between the placement model and the empirical rather than the simulated stats of an SDN environment.

This paper focuses on the SDN controller placement problem in virtualized network environments, and proposes a model based on deep reinforcement learning to minimize switch-controller (OpenFlow) latency by solving the optimal placement strategy that adapts to dynamically changing network state. The contributions of this paper are: it improves exiting dynamic controller placement studies by considering the discrepancy of network states before and after controller placement; it formally formulates the SDN controller placement problem using deep reinforcement learning in a virtualized network environment, where both Open Flow and Non-Open Flow traffic exist in the network between switches and SDN controller; and it is a novel work that provides tooling and detailed instructions on how the proposed model can be implemented in real-world software-defined networks.

The remaining of this paper is outlined as follows. Section 2 reviews existing studies on latency-aware SDN controller placement problems. Section 3 presents the problem setting. Section 4 provides technical details of proposed strategic controller placement model. Section 5 describes main procedures of experiment setup and experimental results analysis. Section 6 concludes this work and suggests some constructive future extensions.

2. RELATED WORK

There are three primary subproblems in SDN controller placement studies: the number of controllers, the location of controllers, and the switch-controller mapping [4] [5] [6]. In order to optimize controller placement, existing works consider various objectives, for example, the control latency or switch-controller latency, network resilience or controller failure tolerance, deployment cost, load-balancing and quality-of-service [6] [7] [8], and the use of appropriate metrics to evaluate their placement strategy. Since the objective of the proposed controller placement strategy in this paper is to minimize switch-controller latency which is an important factor in the agility of SDN [8] [9] [10] [11], this section reviews existing papers on the latency-aware controller placement problem (CPP) first, then demonstrates the contributions of the proposed model.

Early studies on latency-aware CPP focus on static placement of SDN controller(s). Such works consider placement strategy in the network provisioning phase and formulate CPP as an

optimization problem. Then, classic models and algorithms for solving the facility location problem [12] are used to solve the optimal placement of SDN controller.

The authors of [5] consider average-case control latency and worst-case control latency in their objective function respectively, and formulate it as a minimum *k*-means optimization problem, and a minimum *k*-center optimization problem. However, an enumeration of all possible placements given *k* number of controllers is needed in that paper for solving the optimal placement which is not efficient, because the computation complexity increases exponentially as *k* increases. [8] studies the multi-controller placement problem to minimize switch-controller latency by proposing an optimized *k*-means clustering algorithm, which progressively partitions the network. Their experiments showed that the optimized *k*-means algorithm achieves preferable locally minimum latency. However, the experimental networks used for evaluation are simulated networks and it is assumed that only switch-controller traffic exists in the network, for example, if the SDN environment is based on OpenFlow, then there is only OpenFlow traffic in the network. Most importantly, they use algorithms from graph theory and all nodes in the graph are identical meaning that all nodes are valid locations for switches and controllers, which is not exactly true in real-world networks where controller nodes and switch nodes are heterogeneous. This is also the problem of most SDN controller placement studies using *k*-means. [11] also considers the capacity of SDN controllers when deciding placement to minimize switch-controller latency. The authors use the Integer Programming algorithm proposed in [13], but their solution is still a static placement.

Since the network where switches and SDN controllers operate is always changing, a static controller placement strategy is not optimal in the long term because it lacks elasticity to match the changing network. Therefore, dynamic controller placement that can adapt to network changes has become an important topic in the area of CPP. [14] [15] study dynamic SDN controller placement to respond to the number of new flows generated by switches. In their proposed model, a threshold on the latency between any two nodes in a network is used to partition the network first, then the location of controller(s) is calculated to minimize the maximum switch-controller latency in each sub-network. However, it assumes that all nodes in the network are valid locations for both switches and controllers, and re-computation of optimal controller placement once the network changes is not efficient.

The authors of [1] [16] [17] study dynamic containerized SDN controller placement, and apply a rule-based decision tree to find the optimal controller location given the OpenFlow latency of each switch-controller pair. However, the latency used for running the decision tree to instruct controller placement is recent but outdated data, which could lead to sub-optimal placement.

[18] [19] apply deep reinforcement learning (RL) which is widely used in control systems to solve optimal SDN controller placement. [19] solves dynamic optimal clustering of SDN switches and controllers, but the way the authors define and use states in their model is a misuse of RL. In their model, states are able to be optimized just like actions, but in RL states are a characterization of the environment which can be observed by the agent, then the action of the agent further affects the evolution of the environment. [18] solves dynamic optimal controller locations by considering both the switch-controller latency and load-balancing in their objective function. However, there is only switch-controller traffic in their experimental network, and the data used for training the proposed model is from a simulated network which is different from empirical data. Therefore, their proposed model has limited applicability to virtualized SDN controller scenarios.

In summary, existing studies on CPP have provided both static and dynamic solutions by considering various objectives and constraints of interests, but there are still some open

challenges. First, the static placement strategy cannot adapt to network changes, thus making it less optimal. Second, existing dynamic placement studies have unresolved problems, for example, they assume that the physical network between switches and controllers only transfers switch-controller traffic (such as OpenFlow); simulated networks rather than real software-defined networks are used in experiments; sub-optimal placement is caused by using outdated network information for decision making and misuse of mathematical models in existing learning-based studies. In this paper, a dynamic SDN controller placement strategy based on deep reinforcement learning is proposed to address some of these issues.

3. PROBLEM SETTING

This paper focuses on minimizing the latency of OpenFlow (OF) traffic in a virtualized network environment, where switches communicate with SDN controllers hosted on servers and those servers will also receive additional traffic to their services such as HTTP traffic to a web server hosted in the same virtualized environment. The latency of OF traffic is influenced by various factors, including the available bandwidth of links in the network, which is dynamic due to the changing sizes of both OF traffic and non-OpenFlow (NOF) traffic. Additionally, the location of the active (in use) SDN controller impacts the paths that OF traffic takes, and congested links on these paths may cause delays. While a network orchestrator cannot determine the size and type of traffic initiated by end hosts, it can strategically determine the placement of the active SDN controller to affect network performance. The proposed model introduces an additional layer of granularity to standard server or virtual machine placement which typically prioritizes resource usage of machines over network metrics. In other words, the proposed model enables dynamic service-level relocation, even when server locations are fixed, to improve network performance.

Suppose there are n servers hosting SDN controllers and only one of the them is active each time, using a fixed strategy to place the active SDN controller is usually not optimal because it cannot adapt to changing network state. Therefore, it is desired to adjust the location of the active SDN controller based on the recent network state. The network state characterization in the paper includes the size of both OF traffic and NOF traffic received by each server and the available bandwidth of each path from OF switches to servers. To decide the optimal placement of the active SDN controller, a strategic placement model is proposed based on deep reinforcement learning which is known for solving sequential decision-making problems [20]. Differentiating from exiting studies whose placement strategy is adjusted only according to a single network state, the proposed model considers the inter-dependencies between network states and it makes decision not only by current network state (OF and NOF) but also the effect of a decision on future network states. By considering a long-term reward of reducing the latency of OF traffic, the proposed model is able to make optimal and proactive decision of SDN controller placement, which is beneficial to a tightly-coupled network system.

4. STRATEGIC PLACEMENT MODEL

The proposed SDN controller placement model is based on deep reinforcement learning which has been widely used for solving strategic decision-making problems in control systems [21], such as its applications in AlphaGo and robotics. The mathematical formulation of reinforcement learning is based on the Markov Decision Process [22] whose components can be represented by a 5-tuple (S, A, R, P,γ) in a single agent scenario. In a single agent RL model, the agent interacts with an environment and takes actions to maximize its utility. S is a set of states of the environment. A is a set of feasible actions that the agent can take in each state. R is a set of reward (feedback) received by the agent after taking an action in an environment state. P is a state transition model representing the probability of transitioning from one state to another state

by taking an action, which is usually unknown. γ is called a discount factor which is used to characterize the diminishing effect of future rewards on the value function of current state.

The objective of the agent in the reinforcement learning model is to maximize a long-term utility. So, when the agent decides the action to take in a state, it considers not only the immediate reward but also a sequential of future rewards. Suppose s_t is the environment state at time t, $r(s_t)$ is the immediate reward the agent receives in s_t, and s_T is a terminal state representing the end of a sequence (or an episode), then the long-term utility of the agent in s_t is a cumulative value $V(s_t)$ formulated as follows:

$$V(s_t) = r(s_t) + \gamma r(s_{t+1}) + \gamma^2 r(s_{t+2}) + \cdots + \gamma^{T-t} r(s_T) \quad (1)$$

where γ is the discount factor, a value in [0, 1], $V(s_t)$ is also called the value function, and the objective of the agent is to find the optimal action to maximize it in all environment states. Another important function in reinforcement learning is the state-action value function $Q(s_t, a_t)$ (also known as the Q function). It calculates the long-term utility of the agent when taking the action a_t in environment state s_t:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s'_{t+1}} P(s'_{t+1} | s_t, a_t) \cdot V(s'_{t+1}) \quad (2)$$

where $r(s_t, a_t)$ is the immediate reward of taking the action a_t in environment state s_t, and the second term in Equation (2) is the expectation of future rewards which is the weighted sum of value of all possible states s_{t+1}. Therefore, the optimization function of the agent can be formulated as:

$$\max_{\pi(s)} \sum_a Q(s, a) \cdot \pi(a) \quad (3)$$

where $\pi(s)$ is the agent's strategy of taking actions in state s, which is simply a distribution over all feasible actions. If there is only one action a^* which has the largest Q value in state s, then the optimal strategy will be a fixed strategy, that is, to always take that action a^* in state s.

As stated in the above, in order to characterize the network state, the proposed strategic SDN controller placement model combines the amount of network traffic (both OF and NOF) on the links and the available bandwidth of all paths from OF switches and servers hosting SDN controllers. However, these values in a network state are continuous, and it is impossible to enumerate all possible network states. Therefore, the proposed model is based on deep reinforcement learning which uses a neural network to calculate the optimal placement for the active SDN controller given a network state.

In the proposed model, a scenario with one OF switch and n servers (s_1, s_2, ..., s_n) hosting SDN controllers is considered. The 5-tuple (S, A, R, P,γ) of the model is defined as follows:

State: a network state is represented by (c_i, of_1, nof_1, abw_1, of_2, nof_2, abw_2, ..., of_n, nof_n, abw_n), where c_i ($I \in [1, n]$) is the current active SDN controller, of_k ($k \in [1, n]$) is the OF traffic received by server k, nof_k is the NOF traffic received by server k, and abw_k is the available bandwidth of the path from server k to the OF switch.

Action: the feasible actions of the decision maker (network orchestrator) in any network state are choosing one of the n servers to host the active SDN controller.

Reward: based on the objective of the proposed model which is to minimize the latency of the OF traffic, it is desired to use a latency-related metric to be the feedback to an action. Since a path with better latency is equivalent to a path with larger available bandwidth, the proposed model uses the available bandwidth of the path from the OF switch to the active SDN controller abw_{of} as the immediate reward because it is easy to measure.

State transition: the state transition model P is assumed to be unknown because a real-world network environment has uncertainties, but P can be estimated from observations.

Discount factor: γ is assumed to be a fixed value in [0, 1], which should be adjusted based on the decision maker's view of how important future rewards are to the agent's utility in any state. In the experiments of this paper, $\gamma = 0.8$.

The proposed neural network model has 5 layers, including the input layer, 3 dense layers and the output layer. Suppose n is the number of servers, then the input of the model is a network state whose dimension is 1 by (1+3n); the output is the estimated utility of taking each feasible action in the given network state whose dimension is 1 by n, hence the action with the largest utility is the optimal placement decision; the three dense layers have dimensions of (1 + 3n) by 50, 50 by 50 and 50 by n respectively, the first two dense layers use the ReLU activation function and the third dense layer uses a linear activation function. The Mean Square Error (MSE) loss function and the Adam optimizer are used in the proposed model.

5. EXPERIMENTS

In this section, the main procedures of setting up the experimental environment to implement and evaluate the proposed strategic SDN controller placement model are first presented. Then, the experiment results are shown and discussed.

5.1. Experiment Network Setup

The network used in the experiment of this work consists of three Ubuntu virtual machines. As shown in Figure 1, "SDN Ctrl (controller) 1 VM" and "SDN Ctrl 2 VM" host multiple services including the Floodlight SDN controller, web service and iperf service, and "Mininet VM" has Open vSwitch (OvS), iperf server and Mininet running on it. GNS3 is used to connect the three VMs shown in Figure2. A simple Mininet topology with one OvS and two end hosts is initiated on the "Mininet VM".

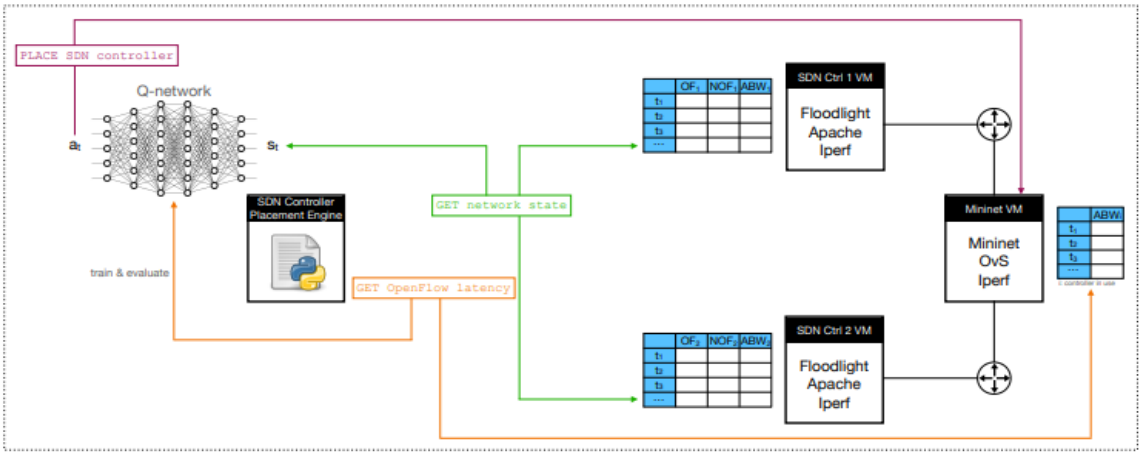


Figure 1.The interactions between the proposed SDN controller placement model and the experimental network

After completing the configurations of all the VMs and network devices in the experimental network to enable connectivity in the network, some network traffic needs to be injected into the network. In the proposed strategic placement model, only two types of traffic are defined in the network state characterization that are OF traffic and NOF traffic, and only the real-time traffic size of OF traffic and the the real-time traffic size of NOF traffic matter. Therefore, Shell scripts which define dynamic*http* request traffic profiles are running on the "Mininet VM" to generate NOF traffic to servers hosting controller and other services. In addition, both the frequency and size of the *http* traffic are manipulated to add variation to traffic patterns. Note that, *iperf* traffic is another source of NOF traffic although the main purpose of using it is to collect available bandwidth of switch-controller paths. Regarding OF traffic generation, shell scripts which define dynamic *ping* request traffic profiles to an unknown IP address are running on the virtual hosts inside the Mininet network, and both the frequency and size of the *ping* traffic are manipulated as well. However, when the destination IP address is unknown, the Packet-In packet to the controller encapsulates an *arp* request instead of the intended *ping* request. To fix that, the MAC address of the unreachable destination is statically added into the MAC address table of each host, then the intended *ping* request encapsulated in the Packet-In packet is successfully forwarded to the controller.

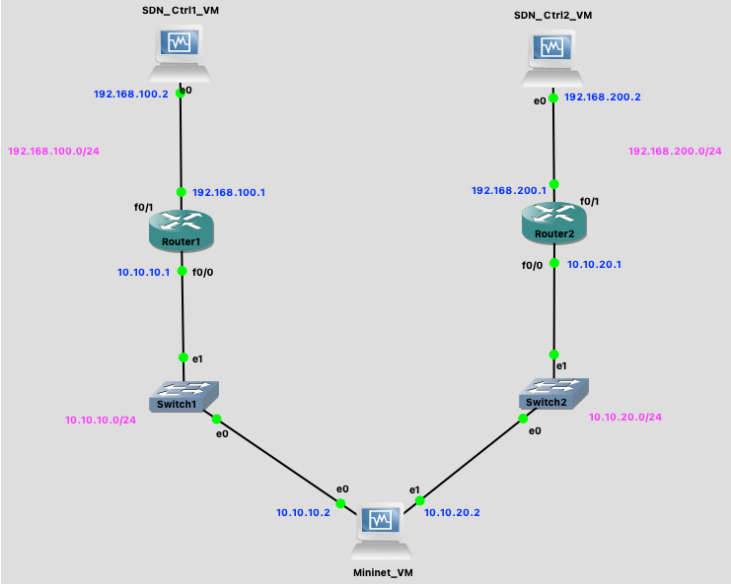


Figure 2. Network topology in GNS3

The proposed strategic SDN controller placement model is hosted on another machine "SDN Controller Placement Engine", which connects to the network via out-of-band communication.

5.2. Network Monitoring and Data Collection

The OF traffic, as well as the NOF traffic received by "SDN Ctrl 1 VM" and "SDN Ctrl 2 VM" is constantly monitored by *tcpdump* and the traffic sizes (in byte) are used to form network state characterization. The available bandwidth of paths from "SDN Ctrl 1 VM" and "SDN Ctrl 2 VM" to "Mininet VM" is constantly measured by *iperf* on controller VMs and is used to form network state.
The available bandwidth of the path taken by OF traffic is measured by the *iperf* on "Mininet VM", because it reflects the OF latency and is used as the reward to current controller placement action.

To extract the information of interest from the output of monitoring data, Python applications are running on the three VMs to constantly process the raw monitoring data and save useful information locally as indicated by the tables in Figure 1.

5.3. Placement Decision-Making and Execution

In order to accelerate the training process of the proposed SDN controller placement model, the deep Q-network (DQN) algorithm [23] is used in this paper. Two characteristics make DQN an efficient deep reinforcement learning algorithm. First, it uses another neural network called "target network" with the same structure as the "Q-network" to calculate the predicted long-term reward quickly, and the weights of the target network are slowly copied from the Q-network without adding training overhead. Second, it maintains an inventory called "replay memory" storing past experiences. Each experience is represented by a 4-tuple (s_t, a_t, r_t, s_{t+1}) telling the immediate reward r_t and the next states$_{t+1}$ after taking the action a_t in the state s_t. The pseudocode of the DQN algorithm is shown in Algorithm 1.

Algorithm 1. DQN for SDN Controller Placement

Data: past experiences (network state, SDN controller placement action, available bandwidth of the OpenFlow path, next network state), represented as (s_t, a_t, r_t, s_{t+1}) stored in replay memory
Input: agent, environment, batch size (20), number of episodes (30), optimizer (Adam)
Output: predicted long-term reward of taking every feasible placement action, $Q(s, a, \theta)$ for $a = 1,..., n$
Initialization: Q-network $Q(s, a, \theta)$, target network $Q(s, a, \theta')$
for $e \leftarrow 0$ **to** 30 **do**
  environment reset s; terminate = false;
  while *terminate == false* **do**
    optimal placement action $a \leftarrow Q(s, a, \theta)$ with decayed ε-greedy;
    configure network to relocate active SDN controller based on action a;
    observe available bandwidth of the OpenFlow path r and next network state s';
    store (s, a, r, s') in replay memory;
    $s = s'$;
    if *terminate == true* **then**
      $\theta' = \theta$
      break;
    else
      sample a batch B from replay memory;
      calculate loss $L = \mathbb{E}[(r + \max_{a'} Q(s', a', \theta') - Q(s, a, \theta))^2]$;
      fit $Q(s, a, \theta)$;
    end
  end
end

The proposed strategic placement model enables the "SDN Controller Placement Engine" to output the optimal placement action given a network state. The code for implementing the proposed model is based on the single agent reinforcement learning setting, which mainly defines an agent class and an environment class with some key functions in each of them.

Key functions performed by the agent are: (1) build the neural network that predicts the long-term reward of taking every placement action in a given network state; (2) extract the optimal placement action from the neural network output, with/without the consideration of decayed ε-greedy in training/testing; (3) train the "Q-network"; (4) maintain and update parameters needed for running the algorithm in the model training process, for example "replay memory" and weights alignment of the "target network"; (5) collect the long-term reward of using different placement strategies.

Key functions performed by the environment are: (1) provide the current network state to the agent, which includes fetching current of_k, nof_k and abw_k ($k \in [1, n]$) from each controller VM; (2) provide the immediate reward to the agent's most recent placement action, which is obtained by fetching the abw_i (i represents the controller in use) from "Mininet VM"; (3) evolve network state according to the agent's action, by converting the optimal placement action from the model to real network configurations to relocate active SDN controller.

5.4. Experiment Results and Analysis

The weights of the Q-network are saved periodically during training, which is beneficial for both version control and iterative and progressive training. To evaluate the performance of the Q-network, the best profile of weights is loaded onto the Q-network and the agent places the SDN controller dynamically by following the output action of the model. Since the Q-network is already trained, no large computation overhead is caused when mapping to the optimal action given a network state.

The long-term average latency of OpenFlow traffic is used for model evaluation. The proposed controller placement strategy which is also simplified as DQN strategy is compared with the following two strategies:

Random strategy: when the agent receives the network state information, it selects the location to place the SDN controller randomly. This random strategy can relate to latency-unaware placement in scenarios where minimizing the switch-controller latency is not part of the objective of the decision maker. For example, when a hypervisor places VMs or VNFs based on resource usages, it is possible that the placement strategy is unrelated to the switch-controller latency.

Generic strategy: when the agent receives the network state information, it only looks at the available bandwidth of paths from each controller server to the switch and picks the server with the largest available bandwidth value to place the SDN controller. It is obvious that this strategy performs poorly when the network states, both before and after controller relocation, are vastly different.

The performance of each of the three placement strategies is shown in Figure 3. The x-axis denotes DQN strategy, generic strategy, and random strategy from left to right. The y-axis indicates the time needed to transfer 1 Kb OpenFlow traffic between switch and controller. The data points used for plotting the box of each strategy are collected as follows. In the proposed controller placement model, 10 consecutive placement actions are defined as one episode, and the average OpenFlow latency in each episode is calculated by dividing the cumulative total latency by 10 which forms one data point. Each strategy is evaluated by 20 episodes and the 20 data points are used in the plot of each strategy in Figure 3.

Three important measurements in a box plot: first quartile Q1, the median and the third quartile Q3 are used for comparing the three placement strategies.

In terms of Q1, the value of the DQN strategy is 0.113, the value of the generic strategy is 0.135, and the value of the random strategy is 0.150. The random strategy is least optimal, and the proposed strategy achieves best OpenFlow latency which improves the generic strategy by 16.3% and improves the random strategy by 24.7%.

In terms of median, the value of the DQN strategy is 0.118, the value of the generic strategy is 0.148, and the value of the random strategy is 0.166. The random strategy is the worst, and the proposed strategy achieves the best OpenFlow latency which improves the generic strategy by 20.3% and improves the random strategy by 28.9%.

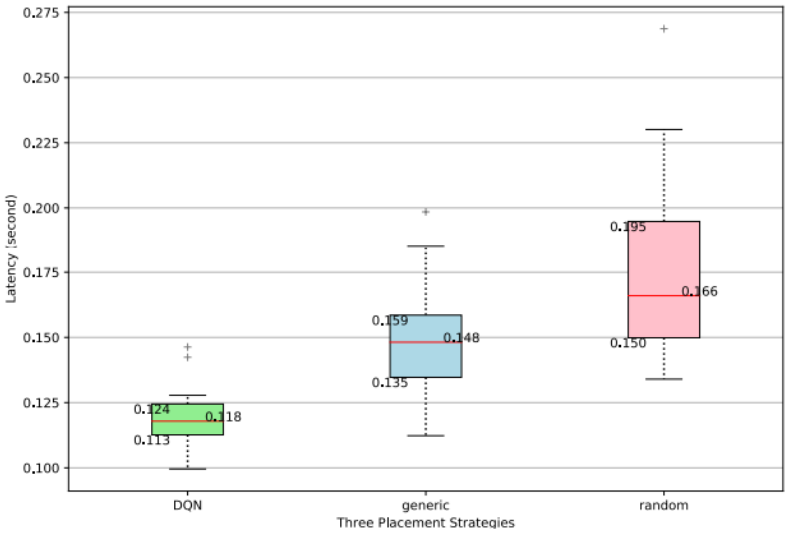


Figure 3. Long-term average latency of transferring 1 Kb Open Flow traffic using different strategies

In terms of Q3, the value of the DQN strategy is 0.124, the value of the generic strategy is 0.159, and the value of the random strategy is 0.195. The random strategy is still the worst, and the proposed strategy achieves the best OpenFlow latency which improves the generic strategy by 22.0% and improves the random strategy by 36.4%.

Therefore, the experiment results show that the proposed SDN controller placement strategy, based on deep reinforcement learning, outperforms the other two placement strategies in terms of minimizing the long-term OpenFlow latency in a dynamic operational network. In addition, as the network expands or the uncertainty of network states increases, the advantage of the proposed SDN controller placement strategy should be more significant.

6. CONCLUSION

This paper proposes a novel dynamic SDN controller placement model, which utilizes deep reinforcement learning techniques to minimize long-term OpenFlow latency. The proposed model effectively addresses several limitations in previous studies on the SDN controller placement problem. These limitations include: the inability of the placement solution to adapt to dynamically changing network states; sub-optimal placement due to the neglect of the discrepancy of network states before and after controller relocation; insufficient tooling and instructions for implementing a dynamic controller placement model in real-world software-defined networks; as well as exclusive switch-controller traffic consideration in works that use deep reinforcement learning.

The main contributions of this paper are as follows. First, it formulates the SDN controller placement problem using deep reinforcement learning in a virtualized network environment, where both OpenFlow and Non-OpenFlow traffic exist in the network between switches and SDN controller. Second, it enhances current dynamic controller placement models by accounting for the discrepancy of network states before and after controller placement, which is handled by the state transition estimation in the model. The experimental results demonstrate that the proposed strategy outperforms both a random strategy and a generic strategy in terms of OpenFlow latency. Additionally, it provides detailed tooling and instructions on how to implement the proposed model in real-world software-defined networks.

The proposed model can also be extended to scenarios that either consider more and conflicting metrics in the objective function, or focus on optimal strategy to place multiple services instead of only the SDN controller. These extensions are left for future works.

REFERENCES

[1] D. Gedia & L. Perigo, (2019) "Latency-aware, static, and dynamic decision-tree placement algorithm for containerized SDN-VNF in OpenFlow architectures", In *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 1-7.

[2] M. P. Odini & A. Manzalini, (2016) "SDN in NFV architectural frame work", *IEEE Software Defined Networks Newsletter*.

[3] R. Munoz et al., (2015) "Integrated SDN/NFV management and orchestration architecture for dynamic deployment of virtual SDN control instances for virtual tenant networks", *Journal of Optical Communications and Networking*, Vol. 7, No. 11, pp. 62-70.

[4] T. Hu et al., (2018) "Multi-controller based software-defined networking: A survey",*IEEE Access*, Vol. 6, pp. 15980-15996.

[5] B. Heller, R. Sherwood & N. McKeown, (2012) "The controller placement problem",*ACM SIGCOMM Computer Communication Review*, Vol. 42, No. 4, pp. 473-478.

[6] T. Das, V. Sridharan & M. Gurusamy, (2019) "A survey on controller placement in SDN",*IEEE communications surveys & tutorials*, Vol. 22, No. 1, pp. 472-503.

[7] G. Wang et al., (2017) "The controller placement problem in software defined networking: A survey", *IEEE Network*, Vol. 31, No. 5, pp. 21-27.

[8] G. Wang et al., (2016) "A K-means-based network partition algorithm for controller placement in software defined network", In *2016 IEEE International Conference on Communications (ICC)*, pp. 1-6.

[9] B. Yanet al., (2021) "A survey of low-latency transmission strategies in software defined networking", *Computer Science Review*, Vol. 40.

[10] M. Jammal et al., (2014) "Software defined networking: State of the art and research challenges", *Computer Networks*, Vol. 72, pp. 74-98.

[11] G. Yaoet al., (2014) "On the capacitated controller placement problem in software defined networks", *IEEE communications letters*, Vol. 18, No. 8, pp. 1339-1342.

[12] R. D. Meller & K. Y. Gau, (1996) "The facility layout problem: recent and emerging trends and perspectives",*Journal of manufacturing systems*, Vol. 15, No. 5, pp. 351-366.

[13] F. A. Özsoy & M. Ç. Pınar, (2006) "An exact algorithm for the capacitated vertex p-center problem", *Computers & Operations Research*, Vol. 33, No. 5, pp. 1420-1436.

[14] M. T. I. ulHuque, G. Jourjon & V. Gramoli, (2015) "Revisiting the controller placement problem", In*2015 IEEE 40th conference on local computer networks (LCN)*, pp. 450-453.

[15] M. T. I. ulHuque et al., (2017) "Large-scale dynamic controller placement",*IEEE Transactions on Network and Service Management*, Vol. 14, No. 1, pp. 63-76.

[16] D. Gedia & Levi Perigo, (2022) "Decision-Tree Placement Algorithm for Containerized VoIP VNFs: A Network Management Approach", In *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, pp. 1-5.

[17] D. Gedia, L. Perigo & R. Gandotra, (2020) "Micro-Economic Benefits of Peer-Producing Containerized Network Functions",*Open Journal of Business and Management*, Vol. 8, No. 5, pp. 2285-2302.

[18] Y. Wu et al., (2020) "Deep reinforcement learning for controller placement in software defined network", In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1254-1259.

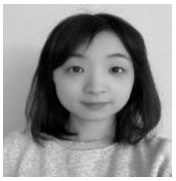
[19] E. H. Bouzidi et al., (2022) "Dynamic clustering of software defined network switches and controller placement using deep reinforcement learning",*Computer Networks*, Vol. 207.

[20] M.V. Otterlo& M. Wiering, (2012) "Reinforcement learning and markov decision processes", *Reinforcement learning: State-of-the-art*, pp. 3-42.

[21] F. Shen, L. Perigo & J. Curry, (2023) "SR2APT: A Detection and Strategic Alert Response Model Against Multistage APT Attacks",In *Security and Communication Networks* (forthcoming), DOI:10.1155/1969/6802359.

[22] M.L. Puterman, (2014) "Markov decision processes: discrete stochastic dynamic programming",*John Wiley & Sons*.
[23] V. Mnih et al., (2015) "Human-level control through deep reinforcement learning",*Nature*, Vol. 518, No. 7540, pp. 529-533.

AUTHORS



Fan Shen received the B.A. degree in Communication from Wuhan University in 2014, and the M.A. degree in Science Communication from the University of Science and Technology of China in 2017. She is currently pursuing the M.S. degree in Network Engineering and the Ph.D. degree in Telecommunications Engineering with the Department of Computer Science, University of Colorado Boulder. Her research interests are cyber intelligence, deep learning, game theory, and Software-defined Networking.

Levi Perigo received the B.S. degree in Information Systems Computer Science from Anderson University, the M.S. in Information and Communication Sciences from Ball State University, and the Ph.D. degree in Information Systems from Nova Southeastern University. He has ten years work experience as a Lead and Senior Network Engineer for ADTRAN Inc., and since 2015 he has been a Scholar in Residence at the University of Colorado Boulder's Professional Master's Degree program in Network Engineering. Dr. Perigo is the Director of the ONF Certified SDN Professional program, on the research council for MEF SDN-NFV, and on the Technical Program Committee for IEEE NFV-SDN.