Received: 4 January 2018 | Revised: 1 June 2018 | Accepted: 20 July 2018

DOI: 10.1002/dac.3802

RESEARCH ARTICLE

WILEY

A congestion control method of SDN data center based on reinforcement learning

Rong Jin[1] | Jiaojiao Li[1] | Xin Tuo[1] | Weiming Wang[1] | Xiaolin Li[2]

[1]School of Information and Electronic Engineering, Zhejiang Gongshang University, Hangzhou, China

[2]Large-scale Intelligent Systems Laboratory, University of Florida, Gainesville, Florida, USA

Correspondence
Rong Jin, School of Information and Electronic Engineering, Zhejiang Gongshang University, Hangzhou, China.
Email: jinrong@mail.zjgsu.edu.cn

Funding information
National Key Research and Development Program of China, Grant/Award Number: 2017YFB0803202; Zhejiang Provincial Key Laboratory of New Network Standards and Technologies (NNST), Grant/Award Number: 2013E10012; National Natural Science Foundation of China, Grant/Award Numbers: 61402408 and 61379120; Zhejiang Provincial Natural Science Foundation of China, Grant/Award Number: LY18F010006; Zhejiang's Key Project of Research and Development Plan, Grant/Award Number: 2017C03058; National High Technology Research and Development Program (863) of China, Grant/Award Number: 2015AA011901

Summary

With the development of cloud computing and big data, the internal communication business in data center has increased dramatically, and then the traffic in data center has also significantly increased. The bandwidth of data center is difficult to meet the bandwidth requirements of those intensive applications, and data center is facing a risk of network congestion. Under the background of the development of network intelligence, software-defined network (SDN) should demonstrate its intelligence as a future network architecture. In this paper, we introduce reinforcement learning into the SDN data center to implement congestion control based on flow. We improve the *Q*-learning and Sarsa algorithms and propose two methods of congestion control based on the algorithms. Test results show that these two congestion control methods can control congestion effectively. And Sarsa method has a better performance of link utilization. The average link utilization of the Sarsa method is 2.4% higher than the *Q*-learning method and is 4.48% higher than the on-demand method.

KEYWORDS

congestion control, data center network, reinforcement learning, software-defined network

1 | INTRODUCTION

With the development of computer networks, more and more colorful businesses are presented to us, and these services require the support of data center. The amount of data that the data center network (DCN)[1] needs to process is getting larger and larger, and an effective congestion control[2] method can ensure the data center to operate normally. Therefore, DCN's congestion control has become one of the hot topics in the field of computer networks.

Software-defined network[3] (SDN) is a new network architecture for future network, the communication business has become more and more busy, and the SDN DCN[4] will face the same congestion problem as the traditional network. The core idea of SDN is the separation of the forwarding plane and the control plane.[5] The control plane has the ability to control and manage the entire network centrally. This facilitates the design of congestion control algorithms for the entire network. Reinforcement learning[6] is developed from the theories of animal learning, stochastic approximation, and optimization control. It is an unsupervised online learning technology. It learns the mapping from the environment

to the action, so that the agent can choose an action from the maximum reward of the environment and make the external environment the best evaluation of the learning system in a certain sense (or the performance of the overall system).

This paper studies the congestion control of the SDN DCN as the starting point, focuses on the congestion control method based on flow, and introduces the reinforcement learning algorithm into the congestion control of SDN DCN. We propose an off-policy congestion control method based on an improved *Q*-learning algorithm[7] and propose an on-policy congestion control method based on the improved Sarsa algorithm.[8] The congestion control methods proposed in this paper not only can avoid network congestion intelligently but also can make the link utilization of the entire network as high as possible. In the experiment, we apply the trained *Q*-matrix to the SDN DCN and compare the on-demand allocation method with our proposed congestion control methods in terms of congestion control effectiveness and link utilization.

This paper introduces reinforcement learning technology into the field of SDN congestion control. The major contributions are described as follows.

1. We introduce reinforcement learning to the network congestion control problem in SDN data centers.
2. Considering the features of the congestion control based on flow in SDN DCNs, we improve the *Q*-learning algorithm and then propose an off-policy congestion control method based on the improved *Q*-learning algorithm.
3. Considering the features of the congestion control based on flow in SDN DCNs, we improve the Sarsa algorithm and then propose an on-policy congestion control method based on the improved Sarsa algorithm.

2 | RELATED WORK

Most congestion control methods for data center congestion control are implemented by adjusting the transmission rate, by setting the priority for flows, or by rerouting.

Mirkovic et al[9] used explicit congestion notification (ECN) to determine the degree of congestion and proposed a data center transmission control protocol (DCTCP) strategy to reduce transmission rate according to the degree of congestion.

Vamanan et al[10] proposed the deadline-aware data center transmission control protocol (D2TCP). The difference between D2TCP and DCTCP is that they use a distributed method to allocate bandwidth, use ECN feedback deadline to control the congestion effectively, and help urgent flows not miss the transmission deadline. Both DCTCP and D2TCP are based on packet.

Alinzadeh et al[11] proposed a concise data center transport protocol, and it is called pFabric. The method adds priority for the flow in advance and schedules flow according to the priority.

Zats et al[12] proposed a new cross-layer network stack in order to reduce the long tail of the process completion time. This method uses cross-layer information to reduce packet loss, gives priority to delay-sensitive flows, and then effectively reduces the transmission time.

Perry et al[13] proposed a congestion method based on flowlet. The idea is to compute optimal rates for a set of active flowlets and update those rates dynamically when flowlets enter or leave the network.

Kanagvelu et al[14] proposed a scheduling method based on flow. They indicate that 85% of flows are smaller than 100 KB in DCNs. Therefore, the flow larger than 100 KB is called an elephant flow, and the flow smaller than 100 KB is called a mouse flow. This method only schedules elephant flow and can reduce the risk of congestion greatly.

The former two methods are based on packet. The latter four methods are based on flow, but they have no intelligence. Software-defined network is based on the granularity of flow, and it should be more intelligent as a future network architecture.[15] In this paper, we introduce reinforcement learning algorithm into the congestion control problem of the SDN DCN. Considering the features[16] of the SDN DCN and the features of the congestion control method based on flow, we improve the traditional *Q*-learning and Sarsa algorithms and propose two congestion control methods for SDN DCN to solve the congestion control problem intelligently.

3 | PROBLEM DESCRIPTION

3.1 | Reinforcement learning model

The model based on reinforcement learning is the process of interaction between an agent and the network environment. The agent can be a controller that controls the source's sending rate. The reinforcement learning model is shown in Figure 1.

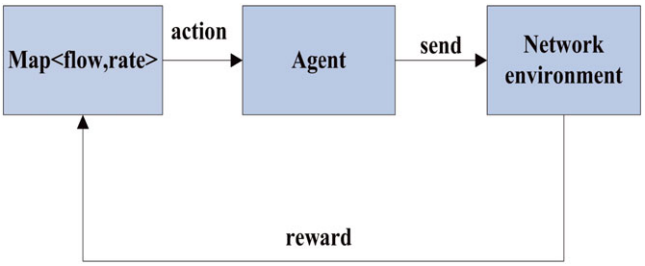

FIGURE 1 Reinforcement learning model

Figure 1 shows the model of reinforcement learning. The controller agent can control the flow rate sent by the source network node; that is, the controller allocates the rate for each flow. The allocation can cause the network environment to change. The network environment is the link bandwidth occupation of all links in the current network. At this time, the agent will obtain reward, the mapping table of flow and rate will be updated, and the agent will select the best action to execute. This cycle will converge to a best mapping table eventually.

3.2 | Problem description

The problem we need to solve is to allocate rates for a set of flows in the DCN. The reinforcement learning model describes the problem where the agent gets a feedback and updates to the next state after performing an action in the current state. This feedback can be called reward. We improve *Q*-learning and Sarsa algorithms. The agent can learn some knowledge after training. The knowledge is represented by a *Q*-matrix.[17] The agent can choose a best action to get the maximum reward from *Q*-matrix in a certain link state. So we describe the congestion control problem in the data center as a five-tuple[18] $<F, S, R, A, Q>$.

(i) F (Flow)

F represents the flows. The queue length of these flows is N, and these flows can be described as (1).

$$F = (flow_1, flow_2, ..., flow_i, ..., flow_N). \tag{1}$$

Assuming there are M links in a given DCN, and flow_i will flow through link j and k, the flow_i can be described as $\text{flow}_i \in \{f_{jk}\}, j, k \in 1, 2, ..., M$.

(ii) S (Link state)

S represents the link state. Here we use link's occupied bandwidth as link state. Suppose that the DCN has M links; the link state is a vector with a length of M. It can be expressed as (2).

$$S = (ls_1, ls_2, ..., ls_i, ..., ls_M). \tag{2}$$

The value of l_{S_i} can be described as $ls_i \in \{g_j\}, j \in 1, 2, ..., P$, where g_j represents the grade of the occupied bandwidth of the link and P is the number of grades. For example, if the link bandwidth is B, we can divide the link states into eight grades as shown in Table 1.

(iii) A (Action)

A represents the action of allocating special rates for special flows according to their link requirements. It can be expressed as (3).

$$A = (a_1, a_2, ..., a_i, ..., a_N). \tag{3}$$

The value of a_i can be described as $a_i \in \{x_1, x_2, ..., x_i, ..., x_L\}$, where L represents the number of possible actions.

TABLE 1 Grades of link state for an example

Link states	Grade
0-B/8	g1
B/8-B/4	g2
B/4-3B/8	g3
3B/8-B/2	g4
B/2-5B/8	g5
5B/8-3B/4	g6
3B/4-7B/8	g7
7B/8-B	g8

(iv) R (Reward)

R represents the rewards after executing action A. In this paper, the reward function is described as (4).

$$R = \min\left(h/7100^{*}(TH-h)\right). \tag{4}$$

Here h represents the occupied bandwidth of the link and TH represents the threshold of congestion. If the occupied bandwidth of the link is less than TH, the reward is obtained by the first function, and higher h can get higher reward. If the occupied bandwidth of the link exceeds TH, the reward is obtained by the second function, and it is negative because the link occurs congestion.

(v) Q (Q-matrix)

Q represents the Q-matrix after training. If the number of link is M and the grade of link state is P, the Q-matrix is a matrix with P^M rows and P^M columns. The initial Q-matrix is a zero matrix. After training, Q value will be updated and converged into a Q-matrix, and it can be described as (5), where q_{ij} represents the updated Q value from state S_i to state S_j.

$$Q = \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1i} & \cdots & q_{1P^M} \\ q_{21} & q_{22} & \cdots & q_{2i} & \cdots & q_{2P^M} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ q_{i1} & q_{i2} & \cdots & q_{ij} & \cdots & q_{iP^M} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ q_{P^M1} & q_{P^M2} & \cdots & q_{P^Mi} & \cdots & q_{P^MP^M} \end{pmatrix}. \tag{5}$$

4 | ALGORITHMS

We describe the improved Q-learning and Sarsa algorithms and use them to avoid congestion for SDN DCN.

4.1 | Improved Q-learning algorithm

In Section 3.2, the congestion control problem of SDN DCN is described as a five-tuple $<F, S, A, R, Q>$, and we introduce the Q-learning to the congestion control of SDN DCN. Considering the features of SDN DCN and the features of congestion control methods based on flow, we improve the traditional Q-learning algorithm, and it is shown as follows (Figure 2).

Firstly, we initialize the reward matrix, we use the reward function described as formula (4), and we also initialize the Q-matrix as a zero matrix. Then we train the Q-matrix for many episodes. For each episode, we set initial state S for

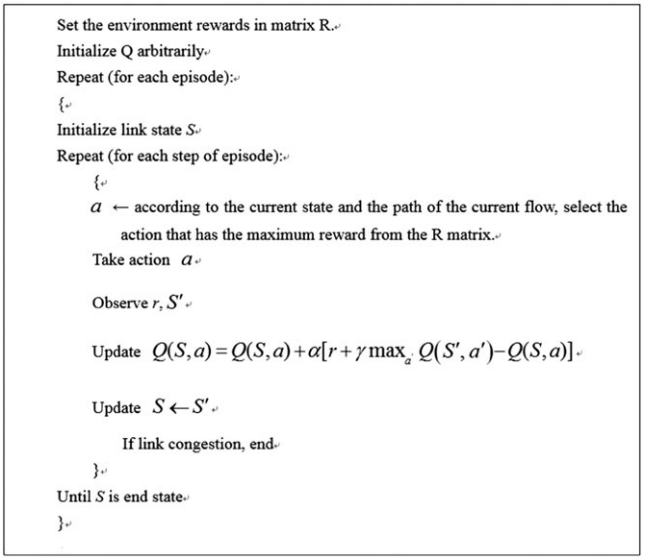

FIGURE 2 The improved *Q*-learning algorithm

the current link randomly, and we update *Q*-matrix step by step. Here an episode refers to the completion of the rate allocation for all the flows, and a step refers to the rate allocation for a flow. For each step, we select the action that has the maximum reward from the *R*-matrix according to the current state and the path of the current flow, execute the action, observe the reward and the new link states, update the *Q* value according to the new reward, and update the current link states. At the end of each step, we check whether the link states are congested. If congestion occurs, end this episode and go to the next episode; if no congestion occurs, go to the next step until *S* is end state. The updating formula of *Q* is described as (6).

$$Q(S,a) \leftarrow Q(S,a) + a\left[r + \gamma \max Q_a\left(S',\ a\right) - Q(S,\ a)\right]. \tag{6}$$

Compared with the traditional *Q*-learning algorithm, the improvement is embodied in two aspects. On the one hand, when selecting action based on the *R*-matrix, the traditional *Q*-learning algorithm selects the action with the maximum reward in all columns of the row corresponding to the current state; but the improved *Q*-learning algorithm must consider the routes of the current flow. The links that the current flow will go through have been determined by routing in advance; when selecting action according to the reward matrix, we cannot select arbitrarily; we should select the maximum reward in several corresponding columns by considering the route of the flow. On the other hand, in every step of an episode, we add the congestion judgment, we judge whether the bandwidth occupancy of the link has reached a threshold, and if the congestion has occurred, the *Q* training episode is stopped.

After training, we will obtain a converged *Q*-matrix eventually. The *Q*-matrix is the knowledge learned by the algorithm. We globally assign the appropriate rate to each flow, then avoid congestion, and realize congestion control. The rate allocation decision is based on the *Q*-matrix trained by the improved *Q*-learning algorithm. We call this congestion control method as off-policy congestion control method based on *Q*-learning. This method can not only make the utilization of the link as high as possible but also avoid the network congestion.

4.2 | Improved Sarsa algorithm

In Section 4.1, we introduce the *Q*-learning to the congestion control of SDN DCN and propose an off-policy congestion control method based on *Q*-learning. The convergence speed of off-policy method is lower than the convergence speed of on-policy method. Therefore, we propose an on-policy congestion control method based on Sarsa. Furthermore, in order to ensure the convergence of Sarsa algorithm, we use greedy in the limit and infinitely exploration (GLIE) policy.[19] So considering the features of SDN DCN and the features of congestion control methods based on flow, we improve the traditional Sarsa algorithm, and it is shown as follows (Figure 3).

Set the environment rewards in matrix R.
Initialize Q arbitrarily
Repeat (for each episode):
{
Initialize link state S
Choose action $a \leftarrow$according to ε-greedy policy, the current state, and the path of the current flow.

Repeat (for each step of episode):
{
Take action a

Observe r, S'

Choose action $a' \leftarrow$according to ε-greedy policy, the current state, and the path of the current flow.

Update $Q(S,a) = Q(S,a) + \alpha[r + \gamma Q(S',a') - Q(S,a)]$

Update $S \leftarrow S'$, $a \leftarrow a'$;

If link congestion, end
}

FIGURE 3 The improved Sarsa algorithm

Firstly, we initialize the reward matrix, we use the reward function described as formula (4), and we also initialize the *Q*-matrix as a zero matrix. Then we train the *Q*-matrix for many episodes. For each episode, we set initial state *S* for the current link randomly, and we update *Q*-matrix step by step. Here an episode refers to the completion of the rate allocation for all the flows, and a step refers to the rate allocation for a flow. For each step, we select the action that has the maximum reward from the *R*-matrix according to the ε-greedy policy, the current state, and the path of the current flow and then execute the action, observe the reward and the new link states, update the *Q* value according to the new reward, and update the current link states and actions. At the end of each step, we check whether the link states are congested. If congestion occurs, end this episode and go to the next episode; if no congestion occurs, go to the next step until *S* is end state. The updating formula of *Q* is described as (7).

$$Q(S,\ a) \leftarrow Q(S,\ a) + a\left[r + \gamma Q\left(S',\ a'\right) - Q(S,a)\right]. \tag{7}$$

Compared with the traditional Sarsa algorithm, the improvement is embodied in two aspects. On the one hand, when selecting action according to the *R*-matrix, the traditional Sarsa algorithm selects the action with the maximum reward in all columns of the row corresponding to the current state; but the improved Sarsa algorithm must consider the routes of the current flow. The links that the current flow will go through have been determined by routing in advance; when selecting action according to the reward matrix, we cannot select arbitrarily; we should select the maximum reward in several corresponding columns by considering the route of the flow. On the other hand, in every step of an episode, we add the congestion judgment, we judge whether the bandwidth occupancy of the link has reached a threshold, and if the congestion has occurred, the *Q* training episode is stopped.

After training, we will obtain a converged *Q*-matrix eventually. The *Q*-matrix is the knowledge learned by the algorithm. We globally assign the appropriate rate to each flow, then avoid congestion, and realize congestion control. The rate allocation decision is based on the *Q*-matrix trained by the improved Sarsa algorithm. We call this congestion control method as on-policy congestion control method based on Sarsa. This method can not only make the utilization of the link as high as possible but also avoid the network congestion.

5 | TEST AND RESULT ANALYSIS

5.1 | Test environment

Our simulation test is based on the VMware11.0.0, Ubuntu 14.04 operating system, and Mininet2.2.1 platform.[20] VMware is a virtual machine software for installing Mininet's Ubuntu virtual image on a Windows system. Ubuntu

is an operating system based on Linux. Mininet is a SDN simulator developed by researchers at Stanford University. It uses lightweight virtualization technology to build a virtual network environment with virtual end points, switches, and routers. Mininet supports OpenFlow[21] and OpenvSwitch protocols. Mininet Provides an API for python, which facilitates collaborative development. Mininet also supports complex topologies and user defined network structures. What is more, it has good hardware portability and high extensibility.

In this paper, we install the VMware virtual machine software in the Window system, install Linux-based Ubuntu operating system under VMware, and then install Mininet in this operating system to create the network topology environment. This paper uses Mininet to customize the DCN topology as shown in Figure 4. The topology includes a core layer switch (S1) and five edge layer switches (S2, S3, S4, S5, and S6).

5.2 | System architecture

To implement congestion control method in SDN data center based on reinforcement learning, we design a system architecture shown in Figure 5. The basic function of each module is described as follows.

1. The perception module is a state detection/processor. It is responsible for collecting the current link states of the DCN.
2. The learning module is an agent *Q*. It is responsible for executing algorithm with the link states information and providing the value of *Q* for the decision-making module.
3. The decision-making module is a policy decision maker. It makes the control policy based on the information provided by the learning module. The policy in this paper is the flow rate allocation scheme.
4. The execution module is a policy executor. It executes the control policy determined by the decision-making module and allocates rate for the flows.
5. The link environment is responsible for maintaining the current link environment.
6. The flow detector is responsible for detecting the arrival of the flows.

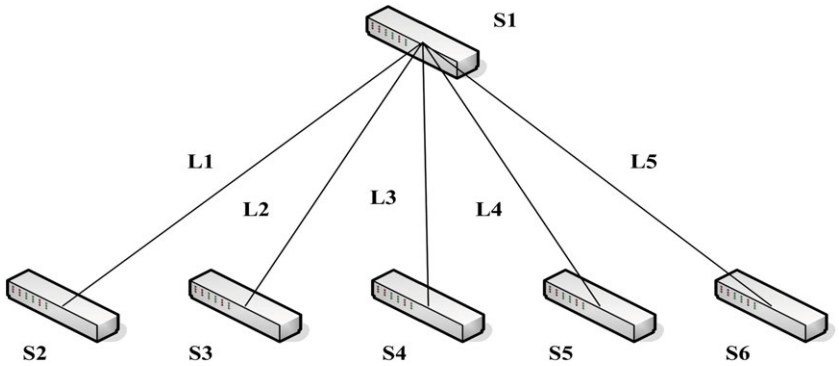

FIGURE 4 The topology of data center

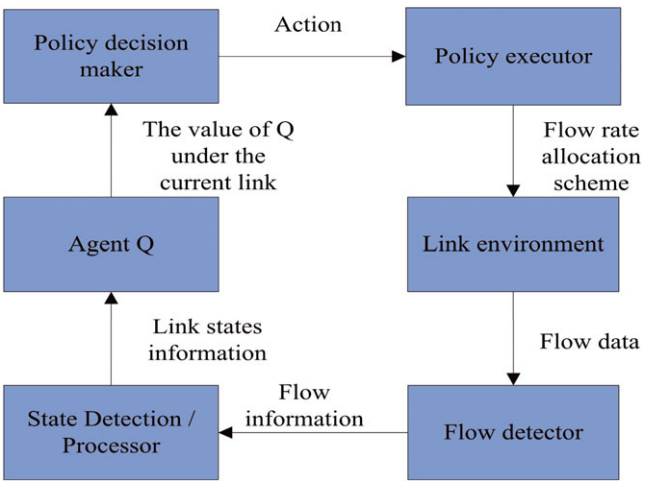

FIGURE 5 System architecture

5.3 | Learning parameters

The setting of parameters in the test directly affects the convergence speed of the algorithm. We use the common values for learning factor γ, discount factor α, and search factor ε of reinforcement learning algorithm. The details of the learning parameters setting are given in Table 2.

6 | TEST RESULT

6.1 | Validation of congestion control

We verify the effectiveness of congestion control based on improved *Q*-learning algorithm and improved Sarsa algorithm by several tests. In this paper, we set the initial occupied bandwidth of the five links as 18G, 20G, 18G, 14G, and 29G. The rate requirement of each flow is 5G. Then we record the changes of the states with the rate allocation for 10 flows.

Figure 6 shows the change of the occupied bandwidth of the five links when we use the on-demand allocation method under the network conditions described above. The on-demand allocation method allocates rate to meet the demand of flow one by one. As shown in Figure 6, after rate allocation for four flows, the occupied bandwidth of link L1 exceeds the threshold, which is 35G. Then network congestion happens. The test shows the on-demand method cannot control congestion very well.

Figure 7 shows the changes of the occupied bandwidth of the five links when we use the off-policy congestion control method based on *Q*-learning under the network conditions described above. From Figure 7, it can be clearly seen that all the occupied link bandwidth does not exceed 35G after rate allocation for all the 10 flows; that is, there is no congestion occurring in the network. The test shows the *Q*-learning method can achieve congestion control by allocating rates for all flows intelligently and globally.

TABLE 2 Parameters setting

Parameters	Values
γ	0.9
α	0.99
ε	0.01
Set of action	{1G, 2G, 3G, 4G, 5G}
Reward function	$\min(h/7100^{*}(35-h))$
Rate requirement of flow	5G
Link bandwidth	40G
Congestion threshold	35G
Initial link state	(18G, 20G, 18G, 14G, 29G)

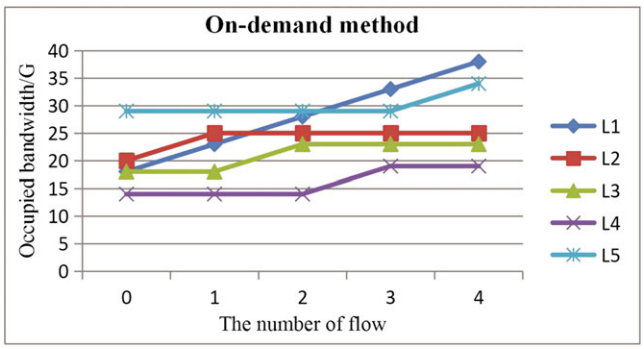

FIGURE 6 The congestion control test for on-demand allocation method

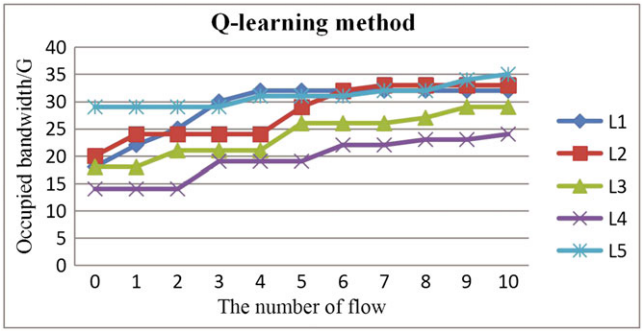

FIGURE 7 Congestion control test for *Q*-learning allocation method

Figure 8 shows the changes of the occupied bandwidth of the five links when we use the on-policy congestion control method based on Sarsa under the network conditions described above. It can also be clearly seen that all the occupied link bandwidth does not exceed 35G after rate allocation; that is, there is also no congestion occurring in the network. The test shows the Sarsa method can also achieve congestion control by allocating rates for all flows intelligently and globally.

In summary, *Q*-learning method and Sarsa method can achieve congestion control intelligently. Test results show the feasibility and effectiveness of the two congestion control methods proposed in this paper.

6.2 | Validation of link utilization

We compare the performances of the three methods by link utilization; we keep the initial state of the links unchanged and test the bandwidth utilization of the links when the bandwidth of the flow increases gradually. We set the initial bandwidth of the five links as 18G, 20G, 18G, 14G, and 29G and change the flows' rate requirements. In this case, we observe the link utilization when using three methods: on-demand allocation method, off-policy congestion control method based on *Q*-learning, and on-policy congestion control method based on Sarsa. Test results are shown in Figure 9.

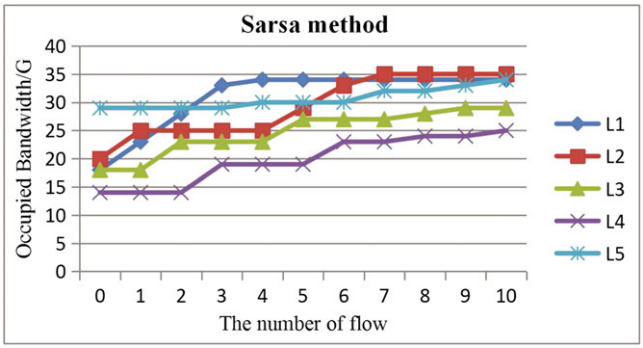

FIGURE 8 Congestion control test for Sarsa allocation method

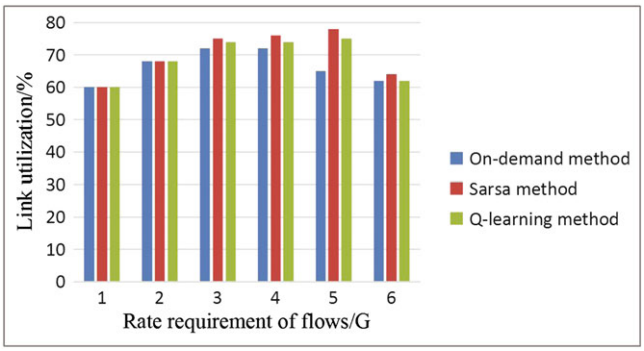

FIGURE 9 Comparison of link utilization of different algorithms under different rate requirements

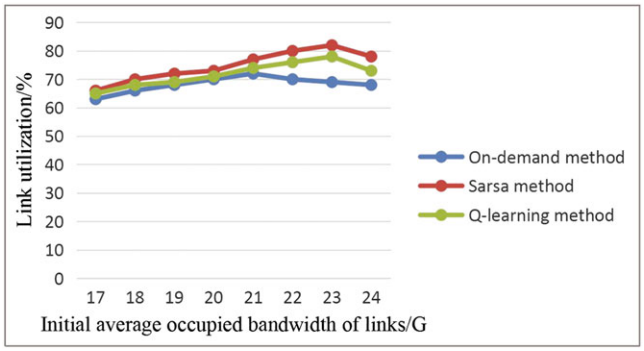

FIGURE 10 Comparison of link utilization of different algorithms in different initial link states

Figure 9 shows that the average link utilization of the Sarsa algorithm is higher than the *Q*-learning algorithm by 4.13% and higher than the on-demand algorithm by 5.63%.

In addition, we keep the rate requirements of the flows unchanged and test the bandwidth utilization of the links when the initial bandwidth of the links increases gradually. Under various network conditions, we observe the link utilization of the entire network under three different methods: the on-demand allocation method, the off-policy congestion control method based on *Q*-learning, and the on-policy congestion control method based on Sarsa. Test results are shown in Figure 10.

Figure 10 shows that the average link utilization of the Sarsa method is higher than the *Q*-learning method by 0.67% and higher than the on-demand method by 3.33%.

In our test scenarios, the average link utilization of the Sarsa method is 2.4% higher than the *Q*-learning method and is 4.48% higher than the on-demand method. In summary, the two congestion control methods proposed in this paper can allocate the flow rate intelligently and globally. They not only can make the link utilization of the network as high as possible but also can avoid the congestion of the entire network. And the on-policy congestion control method based on improved Sarsa algorithm has better link utilization than the off-policy congestion control method based on improved *Q*-learning algorithm.

7 | CONCLUSIONS

With the development of network intelligence, SDN should have the quality of intelligence as a future network architecture. This paper studies the congestion control of the SDN DCN as the starting point, focuses on the congestion control method based on flow, and introduces the reinforcement learning into the congestion control of SDN DCN. We propose an off-policy congestion control method based on an improved *Q*-learning algorithm and proposed an on-policy congestion control method based on an improved Sarsa algorithm. The congestion control methods proposed in this paper not only can avoid network congestion intelligently but also can make the link utilization of the entire network as high as possible.

At the end of this paper, we test the congestion control methods based on *Q*-learning and Sarsa. The results show that these two congestion control methods can achieve congestion control effectively. And Sarsa method has better performance of link utilization. The average link utilization of the Sarsa method is 2.4% higher than the *Q*-learning method and is 4.48% higher than the on-demand method.

We will continue our study in the following aspects in the future.

1. Algorithm optimization. We will take consider the expansion of topology and the optimization of reward functions.
2. Deep reinforcement learning. We will introduce deep reinforcement learning into the congestion control of SDN data center. With the expansion of network topology, *Q*-matrix will become huge. Deep Q-learning Network (DQN) algorithm uses a neural network to replace the *Q*-matrix in *Q*-learning.

ACKNOWLEDGEMENTS

This work was supported in part by a grant from the National High Technology Research and Development Program (863) of China, grant/award number 2015AA011901; the Zhejiang's Key Project of Research and Development Plan,

grant/award number 2017C03058; the Zhejiang Provincial Natural Science Foundation of China, grant/award number LY18F010006; the National Natural Science Foundation of China, grant/award numbers 61379120 and 61402408; the Zhejiang Provincial Key Laboratory of New Network Standards and Technologies (NNST) (no. 2013E10012); and the National Key Research and Development Program of China (no. 2017YFB0803202).

ORCID

Jiaojiao Li http://orcid.org/0000-0002-8418-0517

REFERENCES

1. Barroso LA, Clidaras J, Hoelzle U. The datacenter as a computer: an introduction to the design of warehouse-scale machines. 2009;8(3):154.
2. Cronkita-Ratcliff B, Bergman A, Vargaftik S, Ravi M, Mckeown N. Virtualized congestion control. *Conference on Acm Sigcomm Conference*. 2016;230-243.
3. Foundation O N. Software-defined networking: the new norm for networks. 2012.
4. Bari MF, Boutaba R, Esteves R, et al. Data center network virtualization: a survey. *IEEE Commun Surv Tutor*. 2013;15(2):909-928.
5. Bianco A, Birke R, Giraudo L, Palacin M. OpenFlow switching: data plane performance. IEEE International Conference on Communications. IEEE, 2010:1-5.
6. Sutton RS, Barto AG. Reinforcement learning: an introduction. *IEEE Transactions on Neural Networks*. 2005;16(1):285-286.
7. Watkins CJCH, Dayan P. Technical note: Q-Learning. *Machine Learning*. 992(3-4):79-92.
8. Corazza M, Sangalli A. Q-Learning and SARSA: a comparison between two intelligent stochastic control approaches for financial trading. *Working Papers*. 2015;15(2):1-2.
9. Alinzadeh M, Greenberg A, Maltz DA, Patel J, Prabhakar B, Sengupta S, Sridharan M. Datacenter TCP(DCTCP). In SIGCOMM' 10.
10. Vamanan B, Hasan J, Vijaykumar TN. Deadline-aware datacenter tcp (D2TCP). *Acm Sigcomm Computer Communication Review*. 2012;42(4):115-126.
11. Alizadeh M, Yang S, Sharif M, et al. pFabric:minimal near-optimal datacenter transport. *Computer Communication Review*. 2013;43(4):435-446.
12. Zats D, Das T, Mohan P, Borthakur D, Katz R. DeTail:reducing the flow completion time tail in datacenter networks. *Acm Sigcomm Computer Communication Review*. 2012;42(4):139-150.
13. Perry J, Balakrishnan H, Shah D. Flowtune: Flowlet control for datacenter networks. MIT-CSAIL-TR-2016-011 August 15, 2016
14. Kanagavelu R, Mingjie LN, Mi KM, Lee BS. OpenFlow based control for re-routing with differentiated flows in data center networks. IEEE International Conference on Networks. IEEE, 2013:228-233.
15. Wallner R, Cannistra R. An SDN approach: quality of service using big switch's floodlight open-source controller. *Proceeding of the Asia-Pacific Advanced Network*. 2013;35:14.
16. Foundation O N. Software-defined networking: the new norm for networks. 2012.
17. Torre JDL. An empirically based method of Q-Matrix validation for the DINA model: development and applications. *Journal of Educational Measurement*. 2008;45(4):343-362.
18. Munos R, Stepleton T, Harutyunyan A, Bellemare M. Safe and efficient off-policy reinforcement learning. 2016.
19. Kaur K, Singh J, Ghumman NS. Mininet as software defined networking testing platform. International Conference on Communiction, Computing & Systems. 2014.
20. Mckeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks. *Acm Sigcomm Computer Communication Review*. 2008;38(2):69-74.
21. Consortium O F S. OpenFlow switch specification version 1.0.0. 2009.

How to cite this article: Jin R, Li J, Tuo X, Wang W, Li X. A congestion control method of SDN data center based on reinforcement learning. *Int J Commun Syst*. 2018;31:e3802. https://doi.org/10.1002/dac.3802