2022 IEEE 4th International Conference on Civil Aviation Safety and Information Technology (ICCASIT) | 978-1-6654-6766-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCASIT55263.2022.9987044

Predict Critical Node Pairs Based on Reinforcement Learning in SDN

Haoxian Feng
College of Computer and Information Science
Fujian Agriculture and Forestry University
Fuzhou, China
fenghx@fafu.edu.cn

Zhaogang Shu
College of Computer and Information Science
Fujian Agriculture and Forestry University
Fuzhou, China
zgshu@fafu.edu.cn

Zhifang Zhang
College of Computer and Information Science
Fujian Agriculture and Forestry University
Fuzhou, China
zhangzhifang@fafu.edu.cn

Yuantao Wang
College of Computer and Information Science
Fujian Agriculture and Forestry University
Fuzhou, China
ytwang@fafu.edu.cn

Zhiwei Liu
College of Computer and Information Science
Fujian Agriculture and Forestry University
Fuzhou, China
liuzhiwei@fafu.edu.cn

***Abstract*—Recognizing the node pairs that contribute more packets and bytes compared to others in a Software-Defined Network is essential for many network management tasks, such as traffic engineering, flow-rule placement, anomaly detection. However, recognizing these critical node pairs poses significant challenges because of the enormous solution space for critical node pairs selection and the changing distribution of network traffic. Therefore, it is not reasonable to design heuristic algorithms based on fixed rules to solve this problem because they can not adapt to the changes of the network. In this paper, we propose a reinforcement learning-based framework that a) persistently collects raw network traffic and network configuration information, b) constructs a changing environment and generates reward signals according to the collected data, c) trains the agent periodically to predict critical node pairs dynamically. Our preliminary experiment results show that the proposed framework can predict critical node pairs effectively.**

Keywords—Software Defined Networking, Reinforcement Learning, critical node pairs prediction

I. INTRODUCTION

Software-Defined Networking (SDN) is viewed as one of the most promising methods to address limitations of the current Internet because of its flexibility and programmability [1], [2]. Specially, SDN can supply fine-granularity measurement of traffic flows through the flow rules, which are installed at the switches by utilizing ternary content-addressable memory (TCAM) [3]. However, with the constraint of cost, energy consumption, space, and other factors, the TCAM available in current switches is limited (no more than a few tens of thousands) and it can not come up to the requirements in the real world. When the TCAM becomes full, the switch must evict an old entry before installing a new one. It will result in a significant drop in flow throughput if a switch removes a flow entry that represents an active flow. This situation will be worse if the SDN controller supplies fine-granularity measurement to each flow, because the flow entries will be replaced more frequently. In practice, most of the flows (about 80%) are smaller than 10KB in size, and most of the bytes and packets (more than 80%) are in the top 10% of large flows [4].

In order to design a reasonable flow-rule placement policy, it is necessary to find out which node pairs contribute the most (about 80%) packets and bytes in the network. However, the distribution of network traffic changes all the time [4], so the critical node pairs should be predicted periodically. As presented in Fig 1, from timestamp t_0 to t_1, node pairs 1-7, 3-6, 2-6 contribute the most bytes and packets in the simple network, while other node pairs contribute the rest of bytes and packets. In the next time slot, the critical node pairs changes into 3-6, 2-7, 1-8. In fact, it is not trivial to predict the critical node pairs precisely because of the changing distribution of network traffic and the enormous solution space [9]. Therefore, it is essential to design an effective method that can predict the critical node pairs precisely over time.

In this paper, we propose a reinforcement learning-based framework that can predict the critical node pairs effectively over time. Firstly, it processes the data collected from Data Process Layer (DPL) [5]. Secondly, it makes predictions based on the well-processed data. Though the collected data are always hysteretic, our proposed framework can guarantee the prediction result can be effective for a while.

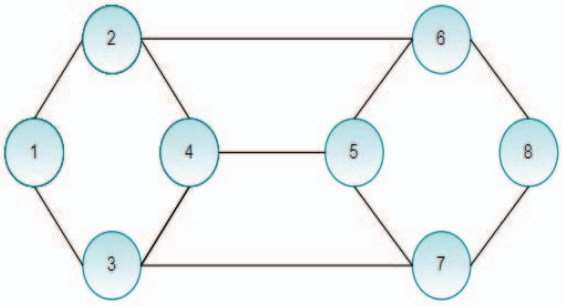
This paper is organized as follows. Section II discussed the related work. Section III presents the framework design. Section IV evaluates the effectiveness of our proposed framework. And the conclusions are drawn in Section V.

II. RELATED WORKS

With the capability of managing networks at a global view and providing network information (e.g., topology information, traffic flow statistics), SDN has attracted great interest from researchers. With the limited TCAM space, it is indispensable to design a reasonable flow rule management policy [3-4]. Shirali-Shahreza *et al.* proposed ReWiFlow [6] to reduce the programming complexity and overcome the hardware limitations through wildcard rules, however, this

978-1-6654-6766-7/22/$31.00 ©2022 IEEE

method would cause loss of visibility due to aggregation of flows. In DeepFlow [7], the author proposed a deep learning-based method to predict the flow size, then adjust the flow-rule placement policy according to the prediction results.



A simple network

Time interval	src	dst	Bytes	Packets
T_1 (t_1-t_0)	1	7	664211615	438734
T_1 (t_1-t_0)	3	6	407082324	286729
T_1 (t_1-t_0)	2	6	517382113	256783
T_2 (t_2-t_1)	3	6	746347270	526294
T_2 (t_2-t_1)	2	7	268516077	189067
T_2 (t_2-t_1)	1	8	205905630	145009

Critical Pairs at each time interval

Fig. 1 Issues about critical node pairs in SDN network. At each time interval, only a very few node pairs can generate the most bytes and packets (over 80%) in the network. The remains are contributed by the rest node pairs

Huang *et al*. proposed a hidden Markov model based method [8] to predict which flows would be maintained over time, which can decrease the number of *Packet-In* packets. Zhang et al. proposed CFR-RL [9], a Reinforcement Learning based method, to discover which source-destination pairs would affect the performance of the network deeply.

However, most of the existing works assumed that the distribution of the network traffic would not change for a long time. That means, once the model is well trained, it does not need to be trained again for a while. In practice, the distribution of the network traffic changes all the time. In this paper, we focus on the active node pairs, and train our proposed model periodically to ensure the performance of prediction.

III. Framework Design

In this section, we present our reinforcement learning-based framework that can periodically predict which node pairs contribute the most (about 80%) bytes and packets in the network.

Inspired by the concept of Knowledge-Defined Networking (KDN) [10], our framework is shown in Fig. 2. It includes three parts, SDN Controller, Data Process Layer, and Network Brain. The functions of each part are discussed as follows:

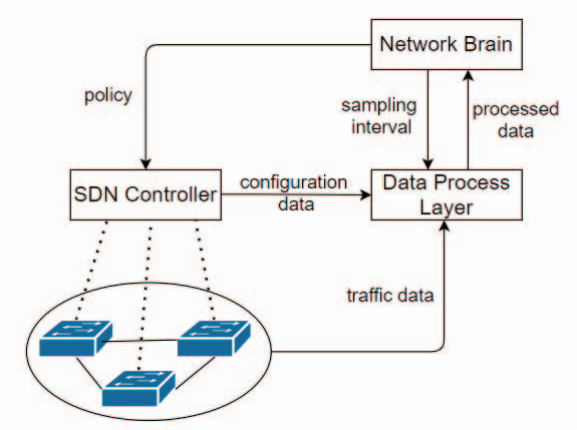


Fig. 2 Proposed framework

A. SDN Controller

SDN controller communicates with the switches through SouthBound APIs to guarantee the normal operation of the network [11]. In general, SDN controller has to process some special packets (e.g., *Packet-In* packet, *hello* packet) that are sent from the switches, and sends the configuration data to the Data Process Layer. What is more, SDN controller executes the policies that are sent from the Network Brain to improve the performance of the network (e.g., lower congestion probability, higher link utilization).

B. Data Process Layer

The responsibility of the Data Process Layer is to collect the raw network traffic data and network configuration data, and send the processed data to Network Brain. We define a node pair P_{T_n} as a set of flows that owns the same source node and the destination node, where T_n is the n_{th} sampling. During each sampling, the Data Process Layer extracts the features of each node pair through analyzing the raw network traffic data and network configuration data, and the extracted features of each node pair are shown in Table I.

TABLE I. Features of Each Node Pair

Feature	Description
Packets	packet count statistic
Bytes	byte count statistics
Time	timestamp of the start of the n_{th} sampling
Duration	duration time in sampling interval

C. Network Brain

Network Brain is the core part of our proposed framework, its responsibility is to send sampling interval T to the Data Process Layer, and send policy to the SDN Controller to improve the performance of the network. In this paper, the main goal of Network Brain is to predict critical node pairs continuously.

In each time slot, the goal of Network Brain can be defined as: *Maximize* $\hat{C}_{T_n} \cap C_{T_n}$, where $\hat{C}_{T_n}$ is a set of predicted node pairs that will be critical during T_n, and C_{T_n} is

a set of real critical node pairs during T_n. Network Brain determines $\hat{C}_{T_n}$ through analyzing the collected data, and C_{T_n} can be calculated through (1).

$$r_{p_i} = \left(\frac{Bytes / Duration / 1024}{2} + \frac{Bytes / Packets}{2}\right) * \frac{Duration}{T} \quad (1)$$

For example, during T_n, Network Brain can sort the node pairs in descending order according to r_{p_i}, and the first K node pairs are critical. However, Network Brain can not calculate C_{T_n} until Data Process Layer finishes the n_{th} sampling, so its goal is to determine a suitable $\hat{C}_{T_n}$ through analyzing the collected data. What is more, the collected data will be larger and larger over time, and it is hard to determine which part of the collected data is valuable and which part is not.

We design a Reinforcement Learning based method in the Network Brain to solve this problem. Each node pair is viewed as a box and the value of the box changes over time. The goal of Network Brain is to select K boxes at each time slot and try its best to maximize the total reward.

To achieve the goal of continuously prediction, we view the prediction task as a game, and we set up many levels for the game. For each level of the game, Network Brain takes two units of collected data as input to construct the environment of the game and generates reward signals. In order to pay more attention to the active connections, we take $s_{n-1} \cup s_n$ as the action set during n_{th} level of the game, where s_n is a set of node pairs that appeared from t_{n-1} to t_n. Then, the Network Brain should take K different actions. We measure whether a selected node pair is good or not based on its statistic of packets and bytes in the past. Once the Network Brain receives new processed data, it updates the game environment, which means that the game enters a new level, then Network Brain retrains itself.

We referred to traditional State-Action-Reward-State'-Action' (SARSA) method [12] for the purpose of training Network Brain effectively, and the quality function is defined as (2), where $v(t_n, p_i)$ represents the total value of p_i from t_0 to t_n, $(r_{p_i})_{T_n}$ is the immediate reward of p_i that can measure the selected node pair is good or not in this level of the game, $\gamma \in [0, 1)$ is the discount factor that determines the importance of future rewards, $\alpha \in [0, 1)$ is the learning rate that determines the override extent of the newly acquired information to the old one.

$$v(t_n, p_i) \leftarrow v(t_n, p_i) + \alpha * [(r_{p_i})_{T_n} + \gamma * v(t_{n+1}, p_i) - v(t_n, p_i)] \quad (2)$$

IV. Experiment Analysis

In order to train Network Brain rationally, we use the real network traffic data which is provided by the WIDE project [13]. The project traces the raw network traffic from 14:00 to 14:15 every day. We collected the raw network traffic data on May 19, 2021. The data volume is 44242MB, and the topology information about their network can be found in [14].

Firstly, we set up sampling interval T to 60 seconds, then we slice the raw data into pieces. Secondly, we capture the statistic information of the flows with the help of a statistical tool in Wireshark [15]. However, as we lack the configuration information of the network, we employ the *GeoIP* tool [16]. We define a node pair as a set of flows that owns the same source ASN number and the same destination ASN number, and the number of node pairs is about 2000 on average. Thirdly, we set up K to 60, which means that at each level of the game, Network Brain can select 60 node pairs.

A. Evaluation

Once we determined the number of critical pairs, we can start to train Network Brain. We set up γ to 0.9 and α to 0.05, then train Network Brain through the first two samples and verify its performance by the 3_{rd} sample. The goal of Network Brain is to maximize the total reward in each level of the game, and our goal is to reach a higher accuracy. We find that the prediction accuracy is highly correlated with the total reward. The result is shown in Fig.3.

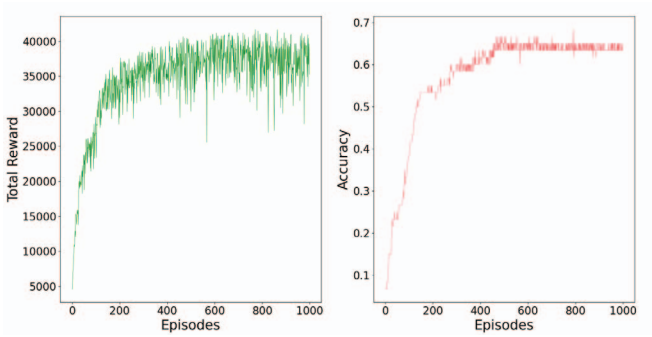


Fig.3 Correlation between total reward and prediction accuracy

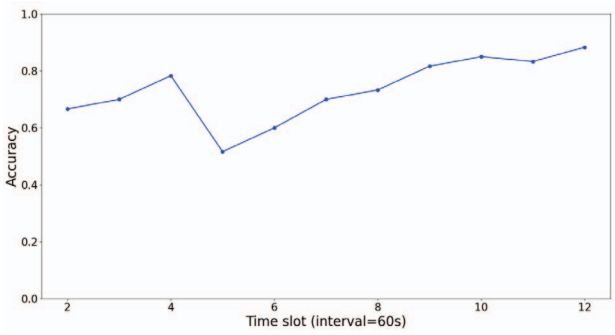


Fig.4 Continuous prediction with a fixed sampling interval

Then, we input the other processed data to the Network Brain, which means that Network Brain makes predictions persistently. The result is shown in Fig.4, and we find that the Network Brain can adapt to network dynamics over time.

B. Comparison

For comparison, we design a rule-based heuristic as the following:

Hard_timeout: Once the critical node pairs are determined at t_n, the Network Brain assumes that these critical node pairs will be maintained during the next fixed time interval T_H. This scheme is simple and similar with the counterpart in [17].

Considering that in the real environment, Data Process Layer needs to spend time collecting and processing the raw network data, so we use the last 5 minutes of network traffic data for performance evaluation. This means that we don't use the last 5 minutes of data for training.

In [8], the author also used 5 minutes of network traffic data for performance evaluation. The comparison result is shown in Table II, and the performance of our proposed framework is better than the *Hard_timeout* scheme. But the prediction accuracy decreased over time, we believe this is caused by the changing distribution of the network traffic. In the real environment, there will be a steady stream of input data, so Network Brain will be smarter and smarter. However, due to the sudden nature of network traffic, how to improve the performance of prediction is still very challenging.

TABLE II. COMPARISON OF PREDICTION ACCURACY

Time(seconds)	60	120	180	240	300
Proposed framework	0.81	0.79	0.78	0.78	0.74
Hard_timeout	0.75	0.75	0.73	0.71	0.7

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed a Reinforcement Learning based framework to predict critical node pairs in the network. The proposed framework can make predictions continuously by adapting to network dynamics. Finally, the performance of our proposed framework is verified through real network traffic data.

In the future, we will investigate how to improve the prediction accuracy quickly if the accuracy is not acceptable in the previous sampling period (e.g., decrease or enlarge the sampling interval). What is more, as each pair consists of many flows, we will investigate the weight of these flows in each critical node pair. And we will redefine the critical pairs in some other situations (e.g., Time-sensitive network) and evaluate the performance of our proposed scheme.

ACKNOWLEDGMENT

This research work has partially received funding from the Fujian Province Natural Science Foundation of China under Agreement (No. 2020J01574).

REFERENCES

[1] Xia W, Wen Y, Foh C H, et al. A survey on software-defined networking[J]. IEEE Communications Surveys \& Tutorials, 2014, 17(1): 27-51.

[2] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks[J]. ACM SIGCOMM computer communication review, 2008, 38(2): 69-74.

[3] Nunes B A A, Mendonca M, Nguyen X N, et al. A survey of software-defined networking: Past, present, and future of programmable networks[J]. IEEE Communications surveys \& tutorials, 2014, 16(3): 1617-1634.

[4] Benson T, Akella A, Maltz D A. Network traffic characteristics of data centers in the wild[C]//Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. 2010: 267-280.

[5] Clemm A, Chandramouli M, Krishnamurthy S. DNA: An SDN framework for distributed network analytics[C]//2015 IFIP/IEEE International Symposium on Integrated Network Management (IM). IEEE, 2015: 9-17.

[6] Shirali-Shahreza S, Ganjali Y. Rewiflow: Restricted wildcard openflow rules[J]. ACM SIGCOMM Computer Communication Review, 2015, 45(5): 29-35.

[7] Lazaris A, Prasanna V K. DeepFlow: a deep learning framework for software-defined measurement[C]//Proceedings of the 2nd Workshop on Cloud-Assisted Networking. 2017: 43-48.

[8] Huang G, Youn H Y. Proactive eviction of flow entry for SDN based on hidden Markov model[J]. Frontiers of Computer Science, 2020, 14(4): 1-10.

[9] Zhang J, Ye M, Guo Z, et al. CFR-RL: Traffic engineering with reinforcement learning in SDN[J]. IEEE Journal on Selected Areas in Communications, 2020, 38(10): 2249-2259.

[10] Mestres A, Rodriguez-Natal A, Carner J, et al. Knowledge-defined networking[J]. ACM SIGCOMM Computer Communication Review, 2017, 47(3): 2-10.

[11] Shin M K, Nam K H, Kim H J. Software-defined networking (SDN): A reference architecture and open APIs[C]//2012 International Conference on ICT Convergence (ICTC). IEEE, 2012: 360-361.

[12] Rummery G A, Niranjan M. On-line Q-learning using connectionist systems[M]. Cambridge, England: University of Cambridge, Department of Engineering, 1994.

[13] Sony C S L, Cho K. Traffic data repository at the WIDE project[C]//Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track. 2000: 263-270.

[14] *Network Topology of WIDE project*.[Online]. Available: http://two.wide.ad.jp/

[15] Orebaugh A, Ramirez G, Beale J. Wireshark \& Ethereal network protocol analyzer toolkit[M]. Elsevier, 2006.

[16] *geoip tool*.[Online]. Avaliable:https://pythonhosted.org/python-geoip/

[17] Zhang L, Lin R, Xu S, et al. AHTM: Achieving efficient flow table utilization in software defined networks[C]//2014 IEEE Global Communications Conference. IEEE, 2014: 1897-1902.