Low Control Latency SD-WANs for Metaverse

Li Qi*, Songshi Dou*, Zehua Guo*, Changlin Li*, Yang Li†, and Tengteng Zhu*

*Beijing Institute of Technology, †China Academic of Electronics and Information Technology

***Abstract*—Metaverse is a shared virtual environment to provide users with an immersive experience and the same feeling as reality. In Software-Defined Wide Area Networks (SD-WANs), the controller dynamically routes flows from metaverse applications to maintain good performance by sending control messages to underlying switches. Thus, the control latency between controllers and switches for rerouting flows is one critical factor for metaverse's performance. However, existing works may introduce high control latency between switches and controllers since they do not consider the dynamic state of each switch in the real world situation. In this paper, we propose to optimize the control latency in SD-WANs by solving a critical programmability-aware problem. Inspired by critical flows, which have a critical impact on network performance, our proposed solution considers the programmability of critical flows at switches to decide the placement of controllers and mapping between switches and controllers. Simulation results show that our work can reduce the control latency by up to 27.5% compared with the baseline algorithms.**

***Keywords*-metaverse, software-defined networking, wide area networks, critical programmability**

I. Introduction

With emerging technologies, such as augment reality, virtual reality, social networks, and digital twins, metaverse has received significant attentions from academia and industry. Many novel devices are proposed to support metaverse. For example, Apple provides users with 360-degree virtual reality rendering by attaching virtual reality headset to Mac [1] and Meta Oculus Quest 2 [2] offers the unique virtual reality experience to users without extra devices. Essentially, metaverse depicts a hypothetical synthetic environment and interacts with the real world [3]. It is independent and parallel to the physical world but can map the features of the real world into it. By using various digital and wearable devices (e.g., augment reality glasses, smartphones, and headsets), users can interact with this synthetic environment. In the metaverse, it is of great significance to provide the realistic feeling and immersive experience for users. However, many factors may affect metaverse's performance. For instance, when virtual objects in the metaverse lag behind the intended position, users would suffer from dizziness and sickness [4]. To promote and develop metaverse, it is very important to guarantee metaverse's performance.

Due to the large and varying computation requirements, metaverse applications are usually deployed in data centers [5] and deliver the service to users via Wide Area Networks (WANs). Thus, WANs play an important role for metaverse's performance. The emerging Software-Defined Networking (SDN) provides new opportunities for maintaining good network performance in WANs, also known as Software-Defined WAN (SD-WAN). For example, Microsoft SWAN [6] and Google B4 [7] show that flexible flow control enabled by SDN can significantly improve network utilization. With the help of high network programmability, SD-WAN can be functioned as infrastructure for metaverse since metaverse applications can be interconnected and executed at different network locations [8]. For large scale metaverse networks, SD-WANs can provide scalable and flexible control over the whole network [9], and also increase the security of metaverse by detecting potential attack behaviors for metaverse applications [10]. As a result. SD-WANs play a critical role in maintaining good performance of metaverse. Typically, an SD-WAN is usually composed of multiple domains, each of which consists of one SDN controller to control flows in the SDN switches placed in its domain. As the network changes, the SDN controller adaptively schedules flows from metaverse applications to maintain their performance by sending control messages to underlying SDN switches. Thus, the control latency between controllers and switches is one critical factor for metaverse's performance. We need to maintain low control latency in SD-WANs for the metaverse.

We face two fundamental problems to maintain low control latency in SD-WANs: (1) where to place the controllers? and (2) how to properly assign switches to controllers? To solve the two problems, Heller *et al.* [11] propose the Controller Placement Problem (CPP), which mainly focuses on placing controllers at proper locations and establishing right switch-controller mapping relationship for minimizing the control latency between switches and controllers. Existing solutions of the CPP are typically coarse-grained location-based. They treat each switch equally and place the controllers only focusing on the distances between locations [11], [12], [13]. In real world, the network state varies, and the number of flows that traverse switches are changing accordingly. Thus, it is not appropriate to simply treat switches at different locations as the same priority because some switches may frequently communicate with controllers while others do not.

In this paper, we propose to optimize the control latency in SD-WANs by rationally placing controllers and establishing switch-controller mapping. The key of the work is to differentiate the impact of switches on the control operation by jointly considering the flow programmability and critical flows since flow programmability is the essential feature of SDN and critical flows have crucial impact on network performance.

The contributions of this paper are summarized as follows:

Li Qi and Songshi Dou contribute equally to this work, and Zehua Guo is the corresponding author.

- We analyze the requirements of the emerging metaverse on SD-WANs and the importance of critical programmability in controller placement and switch-controller mapping for SD-WANs.
- We formulate the Controller Placement and Switch-Controller Mapping (CPSCM) problem as an optimization problem, and solve it to realize low control latency of controllers by properly deciding the controller placement and switch-controller mapping.
- We evaluate the performance of our solution under a real topology with traffic traces. Simulation results show that compared with the baseline algorithms, our solution can reduce the control latency up to 27.5%.

The rest of the paper is organized as follows. In Section II, we introduce the background. In Section III, we introduce the system design of this paper. Section IV mathematically formulates our problem as the CPSCM problem and proposes to efficiently solve the above-mentioned problem. We evaluate and analyze the performance of the proposed critical programmability-aware solution in Section V. Section VI introduces related works, and Section VII concludes this paper.

II. Background

A. Traffic pattern and critical flows

In a network, we do not need to control each flow in the network all the time. We only need to selectively change the paths of some flows to cope with network issues (*e.g.*, congestion), and other flows are always forwarded on their existing paths. A critical flow is defined as a flow with a dominant impact on network performance (*e.g.*, a flow on the most congested link) [14], [15]. Existing works show that critical flows exist in a given traffic matrix [14]. Based on our analysis of real world traffic matrices, our previous work CFR-RL [16] proposes to select critical flows in a network. The critical flows have high traffic load and are likely to experience congestion.

B. Critical flows selection

We train to learn a selection policy over a rich variety of historical traffic matrices, where traffic matrices can be measured by SDN switches and collected by an SDN central controller periodically [17]. The critical flows selection model represents the selection policy as a neural network that maps a "raw" observation (*e.g.*, a traffic matrix) to a combination of critical flows. The neural network provides a scalable and expressive way to incorporate various traffic matrices into the selection policy. Critical flows selection model trains this neural network based on REINFORCE algorithm [18] with customization. Once the training is done, critical flows selection model applies the critical flow selection policy to each real time traffic matrix provided by the SDN controller periodically, where a small number of critical flows are selected.

III. Design overview

In this section, we propose to relieve the controller's control latency of switches by smartly deciding the placement of the controllers, and mapping the switches to the proper controllers.

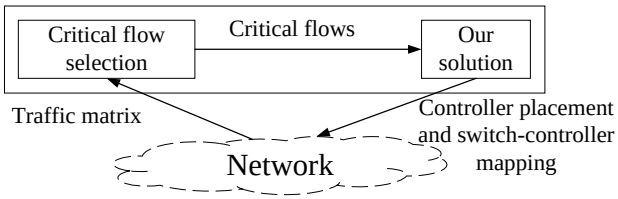


Fig. 1: System Design.

A. System design

Fig. 1 shows the system design of our critical programmability-aware solution, which consists of two modules: Critical Flows Selection, Controller Placement and Switch-Controller Mapping. The first module realizes the selection of critical flows, and the second one realizes optimal placement and mapping. During the network initialization, we generate a set of paths for each source-destination switch pair, and the critical flows selection module periodically gets the information of flows and decides the critical flows based on reinforce learning method. The set of critical flows is sent to the controller placement and switch-controller mapping model.

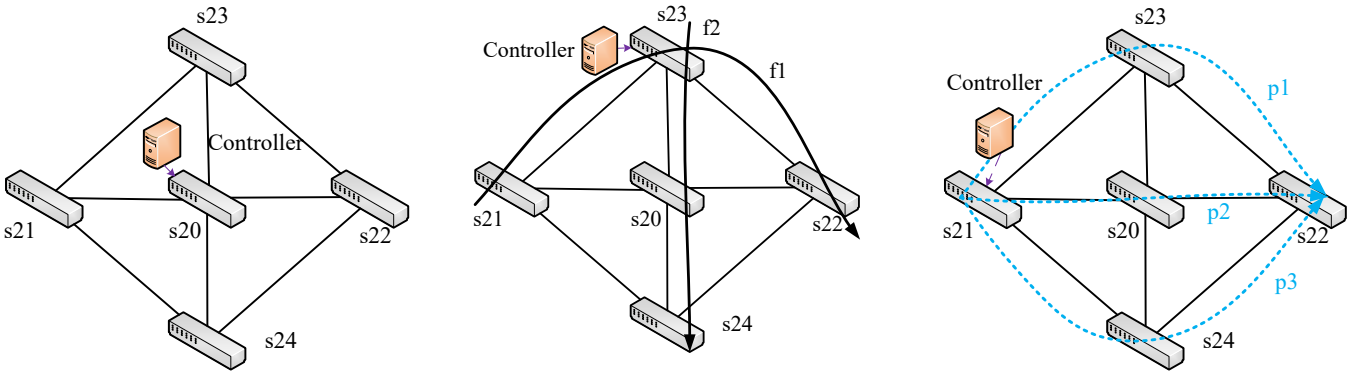
After receiving the set of critical flows, the controller placement and switch-controller mapping model calculates the total number of paths of critical flows that traverse the switch, which is denoted as the total programmability at the switch. Then the controller placement is realized by considering the different priority of programmability at switches and the propagation delay between locations. Switches are mapped to the controllers under the constraint of the limited processing ability of controllers. We formulate the controller placement and switch-controller mapping problem as an optimization problem and detail it in Section IV.

B. A motivation example

We use Fig. 2 to illustrate the design considerations of the controller placement and switch-controller mapping. There are five SDN switches s_{21}-s_{25} and an SDN controller C controls the five switches. Fig. 2(a) shows the result of the solution that only considers the control latency between locations. Controller C is placed at location of s_{20}, which is the middle of this network. However, it is a coarse-grained placement solution because of the network variation and we can not simply treat different switches as the same. Fig. 2(b) considers the influence of flows, and places the controller C at the location of switch s_{23} through which flows traverse more often than other switches. But this solution treats all flows as the same, which is not quite convincing since flows are different in real situation as illustrated above. Fig. 2(c) exhibits our solution, p_1-p_3 are three paths of a critical flow, and the controller C is placed at the location of s_{21} which has the highest programmability of 3.

IV. Problem Formulation and Solution

In this section, we introduce how to optimally place the controllers and map the switches to the controllers by modeling



(a) LocationOnly solution.

(b) FlowNumber-aware solution. Two black lines denote two flows f_1 and f_2.

(c) CriticalProgrammability-aware solution. Three blue dash lines denote three paths from switch s_{21} to switch s_{22}.

Fig. 2: A motivation example.

the system, introducing constraints and the objective function, and formulating an optimization problem. Finally, we propose to solve the CPSCM problem.

A. System description

Typically, an SD-WAN consists of T switches at T locations. Among the T locations, we need to select H controllers at H locations and map the T switches to the H controllers. If the controller is placed at location j, $y_j = 1$; otherwise, $y_j = 0$. We use $x_{ij} = 1$ to denote that switch s_i is mapped to controller C_j; otherwise $x_{ij} = 0$. The network has the set of flows $F = \{f^1, f^2, ..., f^l, ..., f^L\}$. Flows from $[1, M]$ in the set F are critical flows while from $[M+1, L]$ are ordinary ones. For flow $f^l \in F$, its path-set consists of K paths and is denoted as $P_l = \{p_l^1, p_l^2, ..., p_l^k, ..., p_l^K\}$. The set of paths for F is $\mathcal{P} = \{P_1, P_2, ..., P_l, ..., P_L\}$. If path p_l^k traverses switch s_i, we have $\alpha_i^{lk} = 1$, otherwise $\alpha_i^{lk} = 0$. Note that $p_l^{k_0}$ denotes the ordinary path of flow f^l. All the symbols are detailed in Table I.

B. Constraints

1) Switch-controller mapping constraint 1: A switch s_i is strictly mapped to one controller C_j. That is:

$$\sum_{j=1}^{T} x_{ij} = 1, \forall i \in [1, T]. \tag{1}$$

2) Switch-controller mapping constraint 2: A switch s_i can be only mapped to the location where a controller C_j is placed. That is:

$$x_{ij} \leq y_j, \forall i \in [1, T], \forall j \in [1, T]. \tag{2}$$

3) Controller resource constraint: The control load of a controller equals to the total overhead of controlling its associated flows in its domain. We measure a controller's control resource by the number of flows that the controller can normally control without introducing extra delays (*e.g.*, queueing delay [19]). The control load of a controller should not exceed the controller's available control resource and can be written as follows:

$$\sum_{i=1}^{T}(\sum_{l=1}^{L} \alpha_i^{lk_0} * x_{ij}) \leq A_j * y_j, \forall j \in [1, T], \tag{3}$$

where k_0 denotes the original path that flow f traverses, and A_j denotes the control resource of controller C_j.

If we use g_i to denote the number of flows in switch s_i, we can have:

$$g_i = \sum_{l=1}^{L} \alpha_i^{lk_0}, \forall i \in [1, T]. \tag{4}$$

4) Controller placement constraint: We can only place H controllers among T locations. That is:

$$\sum_{j=1}^{T} y_j = H. \tag{5}$$

C. Objective function

The objective is to minimize the control latency of controllers to control the switches that critical flows traverse, which equals to the total propagation delay of the whole paths of critical flows between the switches and their mapped controllers. We use D_{ij} $(D_{ij} \geq 0)$ to denote the propagation delay between switch s_i and controller C_j and formulate the overhead as follows:

$$obj = \sum_{j=1}^{T}\sum_{i=1}^{T}((\sum_{l=1}^{M}\sum_{k=1}^{K} \alpha_i^{lk}) * D_{ij} * x_{ij}).$$

If we use δ_i to denote the total number that the whole paths of critical flows traversing switch s_i, which is also the critical flows' programmability at switch s_i, we can have:

$$\delta_i = \sum_{l=1}^{M}\sum_{k=1}^{K} \alpha_i^{lk}, \forall i \in [1, T]. \tag{6}$$

We can formulate the objective function as follows:

$$obj = \sum_{j=1}^{T}\sum_{i=1}^{T} \delta_i * D_{ij} * x_{ij}. \tag{7}$$

TABLE I: Notations

Notation	Meaning
T	the number of switches
H	the number of controllers
F	the set of flows, $F = \{f^1, f^2, ..., f^l, ..., f^L\}$
M	the number of critical flows
K	the number of paths for flow f^l
s_i	the switch s_i
C_j	the controller C_j
P_l	the path-set for flow f^l, $P_l = \{p_l^1, p_l^2, ..., p_l^k, ..., p_l^K\}$
$\mathcal{P}$	the path-set for F, $\mathcal{P} = \{P_1, P_2, ..., P_l, ..., P_L\}$
α_i^{lk}	a binary constant that denotes if path p_l^k traverses switch s_i
D_{ij}	a constant that denotes the propagation delay between switch s_i and controller C_j
A_j	a constant that denotes the control resource of controller C_j
g_i	a constant that denotes the critical flows' programmability at switch s_i
δ_i	a constant the denotes the propagation delay between switch s_i and controller C_j, which is denoted at Eq. (6)

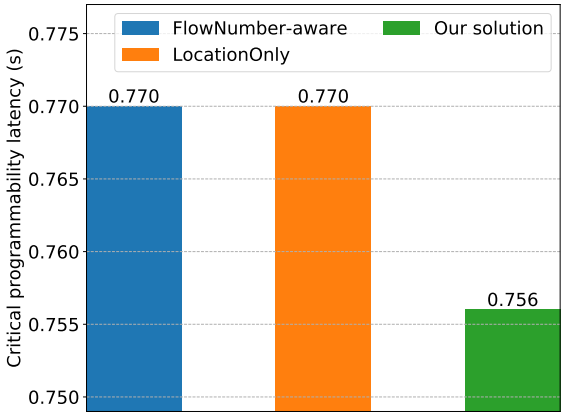


Fig. 3: The results of the critical programmability latency in four weeks. The lower, the better.

D. Problem formulation

The goal of our problem is to minimize the control latency between controllers and switches by selecting the locations of controllers and mapping switches to the placed controllers. Therefore, we formulate this problem as follows:

$$\begin{aligned} \min_{x,y} \quad & \sum_{j=1}^{T}\sum_{i=1}^{T}(\delta_i * D_{ij} * x_{ij}) \\ \text{s.t.} \quad & (1)(2)(3)(5), \\ & y_j, x_{ij} \in \{0,1\}, \\ & \forall i \in [1,T], \forall j \in [1,T], \end{aligned} \tag{P}$$

where $\{\delta_i\}$, $\{D_{ij}\}$, H, and A are constants, and $\{x_{ij}\}$ and $\{y_j\}$ are design variables. A denotes the processing ability of the controller. In this problem, the objective function is linear, and variables are binary integers. Thus, this problem is an Integer Programming (IP).

E. Solution

The typical solution of the above CPSCM problem is to get its optimal result with IP optimization solvers (*e.g.*, GUROBI [20]). However, as the network size increases, the solution space could increase significantly, and finding a feasible solution may cost long time or even impossible. Therefore, the above approach can solve the problem (P) under small size networks. In future works, we will focus on developing an efficient heuristic algorithm for solving the problem (P) to achieve the trade-off between the performance and time complexity.

V. Simulation

A. Simulation setup

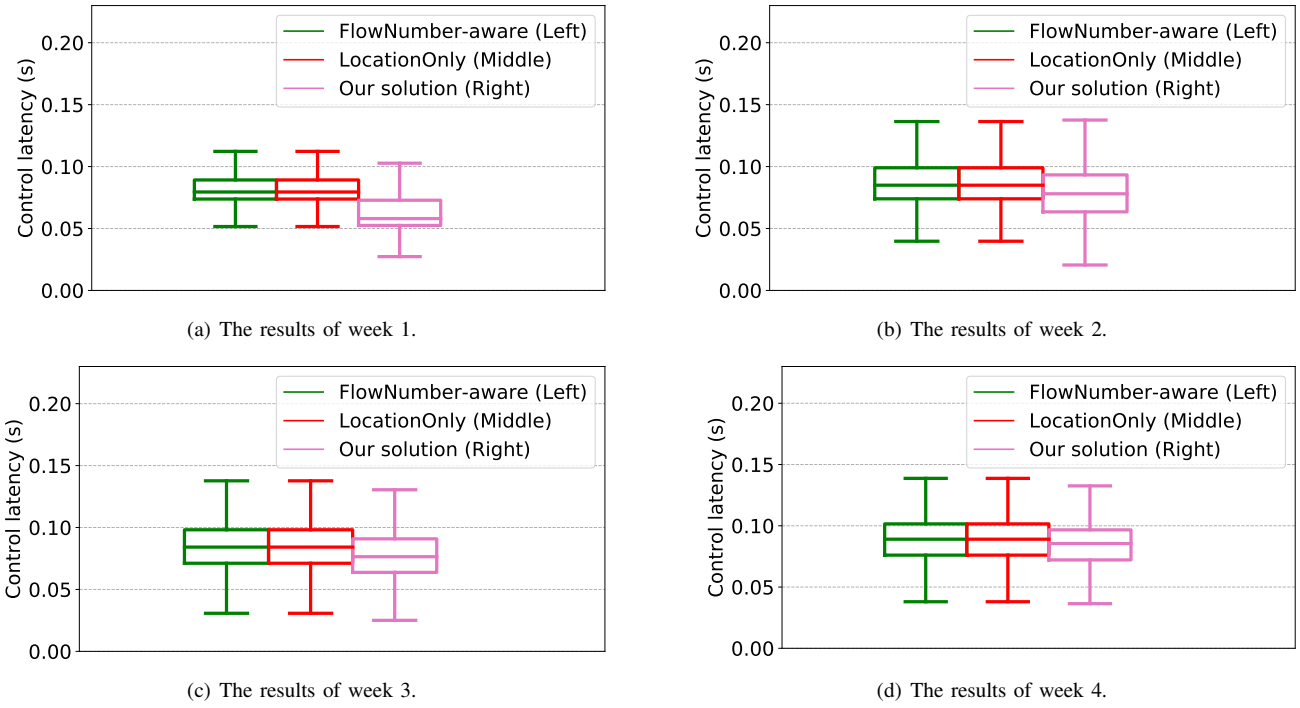
We use a typical backbone topology named Abilene from Topology Zoo [21] to evaluate the performance of our solution. Abilene network is an educational backbone network in North America and consists of 12 switches and 30 links. In Abilene topology, each node has a unique ID and its latitude and longitude. We employ Haversine formula [22] to calculate the distance between two nodes and use the distance divided by the propagation speed (*i.e.*, 2×10^8 m/s) [23] to describe the propagation delay between the two nodes. In our simulation, any two nodes have a traffic flow forwarded on the shortest path. For Abilene, we place three SDN controllers. Following existing works [24], [25], the control resource of a controller refers to the processing ability of the controller to perform flow state pulling operations and get the network state variation without introducing extra control latency [19]. Thus, we set the control resource upper bound of a controller to 500. The dataset records the TMs of 12 switches in a time slot of every five minutes for a duration of six months [26]. We choose 8064 traffic matrices in four weeks from March 29th, 2004 to April 4th, 2004, May 3rd, 2004 to May 16th, 2004, and July 5th, 2004 to July 11th, 2004 as our dataset. For simplicity, we use week 1-4 to represent the traffic matrices in the four weeks.

B. Comparison algorithms

1) FlowNumber-aware: this kind of solutions [27], [28] takes the network state of the number of flows that traversing each switch into consideration to minimize the control latency between switches and controllers.
2) LocationOnly: this kind of solutions [11], [12], [13] only considers the propagation delay between controllers and switches when minimizing the control latency.
3) Our solution: it is the optimal solution of the CPSCM problem. We solve the problem by using GUROBI optimization solver [20].

C. Simulation results

We evaluate our solution in real backbone topology. Achieving load balancing is a common objective in network, and we use GUROBI solver [20] to optimally rerouting flows based on the traffic matrices. When rerouting flows, we need to install flow entries at switches, which emerges control latency between controllers and switches. We calculate the total control latency of all installation during each load balancing



(a) The results of week 1.

(b) The results of week 2.

(c) The results of week 3.

(d) The results of week 4.

Fig. 4: The results of the control latency. The lower, the better.

optimization. Note that the rerouting result of load balancing among all three algorithms are the same, and there are 2016 results for each algorithm in one week.

Fig. 3 shows the results of the critical programmability latency which is also the objective function of CPSCM in four weeks. In the figure, the critical programmability latency of FlowNumber-aware and LocationOnly are higher than our solution. This is because our solution smartly selects the placement of controllers and establish the mapping between switches and controllers to minimize the critical programmability latency. Since the placement of controllers and mapping between switches and controllers calculated by FlowNumber-aware and LocationOnly are the same, the critical programmability latency for them are the same. Since the Abilene topology is small and there are only three controllers in total, it's common that the placement of controllers and mapping between switches and controllers calculated by FlowNumber-aware and LocationOnly are the same. Besides, since the critical flows are the same in four weeks, all three algorithms produce the same result in terms of critical programmability latency in four weeks.

Fig. 4 shows the results of control latency. In Figs. 4(a), (b), and (d), our solution's highest, medium and least are all lower than FlowNumber-aware and LocationOnly, which shows that the fine-grained solution considering the critical programmability at switches performs better than the coarse-grained solution which only considers the propagation latency between different locations. In Fig. 4(a), our solution's medium is 27.5% lower compared with FlowNumber-aware and LocationOnly, while the mediums in Figs. 4(b)-(d) are 7.2%, 9.6%, and 3.6% lower than FlowNumber-aware and LocationOnly respectively. The average control latency of our solution is 11.975% lower than FlowNumber-aware and LocationOnly.

VI. Related Work

For minimizing the control latency, Gao *et al.* [29] define a global latency controller placement problem with capacitated controllers, taking both the latency between controllers and the capacities of controllers into consideration, and propose a particle swarm optimization algorithm to solve the problem. NCPSO [30] is an algorithm which considers the load of controllers, propagation latency and load balancing. Blenk *et al.* [31] introduce mixed integer programming formulations which can optimize the placement of multi-controller switches in virtualized OpenFlow-enabled SDN networks.

As for maintaining load balancing, Killi *et al.* [32] propose a mathematical model for the capacitated controller placement that plans ahead for the failures to avoid a drastic increase in the worst case latency and disconnections. Guo *et al.* [33] take controller's load diversity factor into consideration, and further propose another greedy-based algorithm, which can solve the problem in polynomial time. To keep resiliency and reliability, Beheshti *et al.* [34] analyze resiliency of the connection between control and forwarding planes in SDN. They also propose algorithms to maximize the possibility of fast-failover via resilience-aware controller placement and control traffic routing in the network.

VII. Conclusion and Future Work

In this paper, we propose a critical programmability-aware solution to achieve low control latency in controller placement and switch-controller mapping. The proposed critical programmability-aware solution smartly selects the critical flows and calculates the programmability of critical flows at switches. It realizes fine-grained placement and mapping by considering the different priorities of switches with different programmability.

In the future, we will consider providing rigorous proof of the proposed CPSCM problem's complexity. To efficiently solve the above-mentioned problem when the network increases, we will further develop an efficient heuristic algorithm to achieve the trade-off between the performance and time complexity. Furthermore, we will add more simulations on different topologies to evaluate the scalability of our proposed critical programmability-aware solution.

Acknowledgements

This paper was supported by the National Natural Science Foundation of China under Grant 62002019, the Beijing Municipal Natural Science Foundation under Grant 4222052, and the Beijing Institute of Technology Research Fund Program for Young Scholars.

References

[1] "What is motion?" https://support.apple.com/guide/motion/what-is-motion-motn8d17a294/mac, accessed March 21, 2022.

[2] "Qualcomm. oculus quest 2: How snapdragon xr2 powers the next generation of vr," https://www.qualcomm.com/news/onq/2020/10/29/oculus-quest-2-how-snapdragon-xr2-powers-next-generation-vr, accessed March 21, 2022.

[3] L.-H. Lee, T. Braud, P. Zhou, L. Wang, D. Xu, Z. Lin, A. Kumar, C. Bermejo, and P. Hui, "All one needs to know about metaverse: A complete survey on technological singularity, virtual ecosystem, and research agenda," WorkingPaper, Oct. 2021.

[4] R. L. Holloway, "Registration error analysis for augmented reality," *Presence: Teleoperators & Virtual Environments*, vol. 6, no. 4, pp. 413–432, 1997.

[5] T. Huynh-The, Q.-V. Pham, X.-Q. Pham, T. T. Nguyen, Z. Han, and D.-S. Kim, "Artificial intelligence for the metaverse: A survey," *arXiv preprint arXiv:2202.10336*, 2022.

[6] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 15–26.

[7] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 3–14.

[8] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Compute-and data-intensive networks: The key to the metaverse," *arXiv preprint arXiv:2204.02001*, 2022.

[9] E. H.-K. Wu, C.-S. Chen, T.-K. Yeh, and S.-C. Yeh, "Interactive medical vr streaming service based on software-defined network: Design and implementation," in *2020 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-Taiwan)*. IEEE, 2020, pp. 1–2.

[10] P. Krishnan, K. Jain, R. Buyya, P. Vijayakumar, A. Nayyar, M. Bilal, and H. Song, "Mud-based behavioral profiling security framework for software-defined iot networks," *IEEE Internet of Things Journal*, 2021.

[11] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 473–478, 2012.

[12] P. Yi, T. Hu, Y. Hu, J. Lan, Z. Zhang, and Z. Li, "Sqhcp: Secure-aware and qos-guaranteed heterogeneous controller placement for software-defined networking," *Computer Networks*, vol. 185, p. 107740, 2021.

[13] G. Schütz and J. Martins, "A comprehensive approach for optimizing controller placement in software-defined networks," *Computer Communications*, vol. 159, pp. 198–205, 2020.

[14] J. Zhang, K. Xi, M. Luo, and H. J. Chao, "Dynamic hybrid routing: Achieve load balancing for changing traffic demands," in *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*. IEEE, 2014, pp. 105–110.

[15] J. Zhang, K. Xi, M. Luo, and H. J. Chao, "Load balancing for multiple traffic matrices using sdn hybrid routing," in *2014 IEEE 15th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2014, pp. 44–49.

[16] J. Zhang, M. Ye, Z. Guo, C. Y. Yen, and H. J. Chao, "Cfr-rl: Traffic engineering with reinforcement learning in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2249–2259, 2020.

[17] H. Xu, Z. Yu, C. Qian, X. Li, and Z. Liu, "Minimizing flow statistics collection cost of sdn using wildcard requests," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[18] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[19] J. Xie, D. Guo, X. Li, Y. Shen, and X. Jiang, "Cutting long-tail latency of routing response in software defined networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 384–396, 2018.

[20] "Gurobi optimization," http://www.gurobi.com.

[21] "The internet topology zoo," http://www.topology-zoo.org/, accessed March 21, 2022.

[22] C. C. Robusto, "The cosine-haversine formula," *The American Mathematical Monthly*, vol. 64, no. 1, pp. 38–40, 1957.

[23] "Speed, rates, times, delays: Data link parameters for cse 461," https://courses.cs.washington.edu/courses/cse461/99wi/issues/definitions.html.

[24] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–8.

[25] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "Opentm: traffic matrix estimator for openflow networks," in *International Conference on Passive and Active Network Measurement*. Springer, 2010, pp. 201–210.

[26] "Abilene dataset." http://www.cs.utexas.edu/yzhang/research/AbileneTM, accessed January 21, 2022.

[27] T. Hu, Q. Ren, P. Yi, Z. Li, J. Lan, Y. Hu, and Q. Li, "An efficient approach to robust controller placement for link failures in software-defined networks," *Future Generation Computer Systems*, vol. 124, pp. 187–205, 2021.

[28] P. Yi, T. Hu, Y. Qu, L. Wang, H. Ma, Y. Hu, and J. Lan, "A safe and reliable heterogeneous controller deployment approach in sdn," *China Communications*, vol. 18, no. 8, pp. 47–61, 2021.

[29] C. Gao, H. Wang, F. Zhu, L. Zhai, and S. Yi, "A particle swarm optimization algorithm for controller placement problem in software defined network," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2015, pp. 44–54.

[30] S. Liu, H. Wang, S. Yi, and F. Zhu, "Ncpso: a solution of the controller placement problem in software defined networks," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2015, pp. 213–225.

[31] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control plane latency with sdn network hypervisors: The cost of virtualization," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 366–380, 2016.

[32] B. P. R. Killi and S. V. Rao, "Optimal model for failure foresight capacitated controller placement in software-defined networks," *IEEE Communications Letters*, vol. 20, no. 6, pp. 1108–1111, 2016.

[33] Sheng Guo, Shu Yang, Qi Li, and Yong Jiang, "Towards controller placement for robust software-defined networks," in *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, 2015, pp. 1–8.

[34] N. Beheshti and Y. Zhang, "Fast failover for control traffic in software-defined networks," in *2012 IEEE Global Communications Conference (GLOBECOM)*, 2012, pp. 2665–2670.