Joint Controller Placement and Flow Assignment in Software-Defined Edge Networks

Shunpeng Hua[1] [0009−0001−4139−2579], Baoliu Ye[1] [0000−0003−1065−449X] (✉), Yue Zeng[1] [0000−0002−5553−5534], Zhihao Qu[2] [0000−0001−7538−1985], and Bin Tang[2] [0000−0002−4577−8882]

[1] Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China
[2] School of Computer and Information, Hohai University, Nanjing 211100, China
yebl@nju.edu.cn

Abstract. Software-Defined Networking (SDN) has been introduced into edge networks as a popular paradigm, leveraging its high programmability, where SDN controllers are enabled to centralize network configuration and management. However, frequent flow fluctuations at the edge can result in a large backlog of local flow requests in the processing queue of the controllers, leading to high response delays. Although optimized controller placement and assignment can reduce flow-setup delay, existing approaches are limited in their ability to jointly optimize controller placement and fine-grained flow assignment and address high queuing delay of flow requests. In this paper, we investigate how to jointly optimize controller placement and flow assignment under limited controller capacity, to reduce the propagation delay of data nodes and controllers and the queuing delay of flow requests, and therefore, reducing flow-setup delay. We systematically model the problem and propose a traffic segmentation-based controller placement and flow assignment algorithm. Simulation experimental results demonstrate that our scheme can reduce the flow-setup delay by up to 21.6% compared to existing solutions.

Keywords: Edge computing · software-defined networks · controller placement and flow assignment · latency.

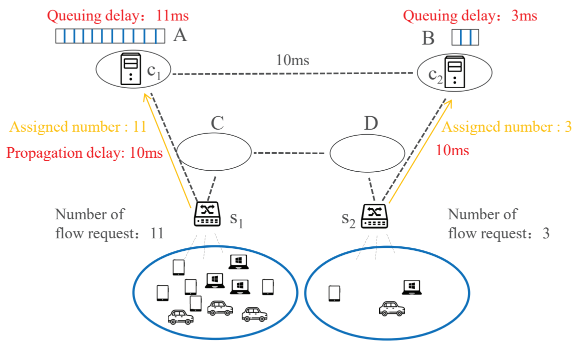
1 Introduction

In recent years, the rise of the Internet of Things has led to the emergence of edge computing as a new computing framework. As an significant technology, edge computing brings computing resources closer to users, resulting in lower latency and reduced energy consumption [1]. As another key technology, Software-Defined Networking (SDN) enables the separation of control and data planes, allowing for centralized control logic on the controller [2, 3]. Thanks to these two technologies, network and computing resources can be flexibly managed and configured [4, 5].

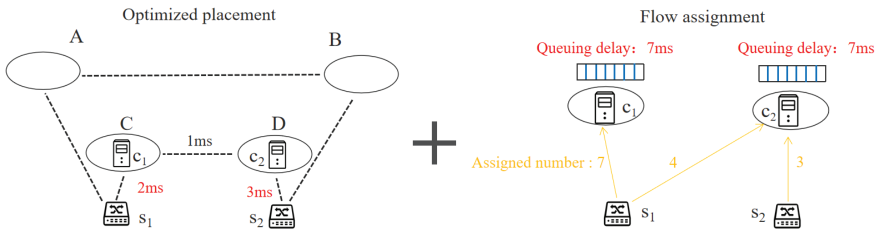
In an SDN-enabled edge network, the control plane and the data plane are decoupled, rather than tightly coupled as in traditional networks. The controllers in the control plane manage the traffic in the network by setting flow rules for the switches in the data plane [6, 7]. The specific process of flow setup is as follows. When a new flow arrives at a switch and there is no corresponding forwarding rule, the switch notifies the controller to plan the routing path by setting flow rules. The controller then computes the forwarding path for that flow and pushes the corresponding flow rules to the switches in the data plane. The aforementioned flow rule setup process introduces flow-setup delay. As networks become larger and more complex, a single controller may not be able to handle the massive flow requests and ensure reasonable flow-setup delay, and a single point of failure can lead to network paralysis. As a result, the control plane is typically implemented with multiple controllers that are logically centralized but physically distributed. Thus, a critical issue is how to coordinate multiple controllers to manage the entire network.

Many works have studied controller placement and switch assignment [4, 8–18], where controller placement refers to which nodes the controllers are placed at, and switch assignment refers to which controllers the switches are assigned to. First, several works [8, 9] have studied switch assignment, which determines which controllers to assign switches to, with a goal of minimizing delay or balancing load. However, edge network is dynamic while static controller placement is lack of flexibility. Therefore, some works [4, 10–18] further study controller placement and switch assignment. Switch assignment is a coarse-grained node-based assignment method compared to flow-based assignment, which bind data forwarding nodes to a certain controller for management over a period of time. The problem with this approach is that if the data forwarding nodes assigned to a certain controller generate a large number of flow requests during a slot, these flow requests will be assigned to the bound controller. At the edge, many nodes have limited computing and storage resources, and the controller's request processing ability is relatively small. This results in a large backlog of requests in the controller request queue, and the controller response delay for these flow requests will be high. Thus, several studies [19, 20] have focused on fine-grained flow-based assignment that determines which controllers to assign the flows to. However, they only consider flow assignment and ignore controller placement, which may lead to inefficient controller plane layout.

Therefore, a critical issue is controller placement and flow assignment. Next, we give an example to show its advantages in detail. As shown in Fig.1, there are four nodes (A, B, C, and D) that can be used to place controllers. As shown in Fig.1(a), the controllers are randomly placed in A and B, which leads to an inefficient control plane layout. In this case, the propagation delay between the controllers and the data plane forwarding devices is as high as 10 ms. Furthermore, the node-based assignment scheme assigns switch s_1 to controller c_1 and switch s_2 to controller c_2. In this case, the queuing delay of flow requests handled by controller c_1 is as high as 11 ms, while controller c_2 remained idle and underutilized. Fig.1(b) shows the optimized controller placement and flow-based assignment scheme, which places controllers on nodes C and D. In this case, the propagation delay between the controllers and the forwarding devices is no



(a) Random controller placement and switch assignment.



(b) Optimized controller placement and flow assignment.

Fig. 1. A motivational example to show the importance of controller placement and flow assignment.

higher than 3ms, which is lower than random controller placement. In addition, flow-based assignment scheme dynamically allocates flow requests of s_1 to the two controllers, which can effectively balance the workload between them. This optimization reduces the queuing delay of the controller processing requests to 7ms, which can improve the efficiency and service performance of the controllers.

Consequently, in this paper, we explore how to dynamically place controllers and assign flows in edge networks to optimize the average flow-setup delay. We present a traffic segmentation-based controller placement and flow assignment scheme to address the high flow-setup delay challenge in edge networks. To the best of our knowledge, this is the first work to combine optimized controller placement and fine-grained flow assignment to tackle the problem of dynamic controller placement and assignment in edge networks. Our proposed scheme is a two-stage algorithm: a) Controller placement, in which we employ traffic segmentation to divide the network into multiple traffic slices with approximately equal traffic loads, and then determine a suitable location for controller place-

Related Work	Decision		Assignment granularity
	Placement	Assignment	Flow-based
[8], [9]	✗	✓	✗
[19], [20]	✗	✓	✓
[4], [10], [11], [12], [13], [14], [15], [16], [17], [18]	✓	✓	✗
Our scheme	✓	✓	✓

ment within each traffic slice, and b) Flow assignment, involving hierarchical flow assignment based on controller positions in the network. The main contributions of this paper can be summarized as follows.

– We formulate the controller placement and flow assignment problem in SDN-enabled edge networks, with the objective of minimizing the average flow-setup delay, which we prove to be NP-hard.
– We introduce a traffic segmentation-based controller placement and flow assignment algorithm to optimize the average flow-setup delay.
– Our proposed scheme outperforms existing approaches in several key indicators, such as average flow-setup delay, percentage of high queuing delay flows, and percentage of flow-setup delay constraint violated flows, as demonstrated by experimental results.

The remainder of this paper is organized as follows. Section II discusses related work and highlights the limitations of these works in SDN-enabled edge networks. We introduce the system model and problem formulation in Section III. In Section IV, we propose a traffic segmentation-based controller placement and hierarchical flow assignment algorithm. Section V presents the evaluation of our proposed scheme.

2 Related Work

Heller et al. [10] first proposed the Controller Placement Problem (CPP) in SDN. Afterwards, a large number of studies on controller placement and assignment emerged. We divide the existing work into three categories: assignment schemes, controller placement and assignment schemes, and controller placement and assignment schemes in software-defined edge networks.

2.1 Assignment Schemes

Sun et al. [8] proposed a dynamic workload balancing scheme based on multi-agent reinforcement learning, which outperformed optimization algorithms to manipulate the mapping relationship between controllers and switches. Huang et al. [9] devised a predictive online switch-controller association and control devolution scheme, which reduced request latency significantly. Xie et al. [19] proposed a light-weight and load-ware switch-to controller selection scheme and designed a general delay-aware switch-to-controller selection scheme to cut the long-tail response latency and provide higher system throughput. Bera et al. [20] proposed a dynamic scheme to assign the flow to the controller to minimize the flow-setup delay and control overhead, which was more fine-grained than the scheme of allocating the switch. This kind of work does not consider the influence of controller position on the system.

2.2 Placement and Assignment Schemes

Guo et al. [11] jointly considered the deployment of SDN switches and controllers to find the locations of updated switches, the locations of deployed controllers, and the mappings between the controllers and the upgraded switches for hybrid SDNs. The authors proposed MapFirst, which returned a high-quality solution with a significant less CPU time. Wu et al. [12] used a deep Q-network to optimize the network delay and load in the dynamic network, optimizing the location of the controller, and dynamically adjusting the mapping between switches and controllers. Basu et al. [13] took the controller placement problem and hypervisor placement problem into consideration at the same time and propose an approach of dynamically deploying controller-hypervisor pairs to provide a variety of network functions with low latency. Bouzidi et al. [14] studied which controllers are selected and enabled in a separate control plane how to partition the set of data plane switches into clusters and assign them to these controllers. The network span studied in the above work is relatively small, and the propagation delay of communication between data nodes and controllers is not fully considered in the decision of controller layout.

2.3 Placement and Assignment Schemes in SDN-enabled Edge Network

Qin et al. [4] placed the controllers in the static edge nodes in SDN-enabled edge networks. The authors took the inter-controller and controller-node overheads as two important factors to formulate the controller placement and assignment problem at the edge and proposed exact and approximate algorithms to optimize delay and overheads. Chen et al. [15] studied the adaptive controller placement and assignment problem to adapt to the dynamic topology and time-varying workload in SDN-based low-earth-orbit satellite networks. The authors proposed control relation graph (CRG) and a CRG-based algorithm which outperforms existing schemes in terms of response time and load balancing. Li et al. [16] proposed a Louvain algorithm-based controller placement policy to solve the CPP in SDN-IoV and a controller replacement policy to adapt to the dynamic topology of IoV. The objectives are to optimize the control load, control delay, intra-cluster delay and throughput. Li et al. [17] proposed an edge service mesh architecture with distributively deployed controllers and investigate the controller placement problem toward communication cost minimization. Soleymanifar et al. [18] introduce two novel maximum entropy based clustering algorithms to address controller placement problem in wireless edge networks.

Table 1. Summary of Notations

Symbol	Descriptions
G	Undirected Network Graph
S	Set of switches
C	Set of controllers
F	Set of flows
P	Placement policy
A	Assignment policy
$\delta_{s,j}$	Propagation delay of the shortest path between the switch s and the controller placed at s_j
$\alpha_j(t)$	Flow-request arrival rate at the controller placed at s_j
μ_j	Execution rate of the controller placed at s_j
w_i	Number of CPU cycles required by the controller to calculate the forwarding path of the flow f_i
D_i	Flow-setup delay of the flow f_i
D	Average flow-setup delay in time slot t
k	Number of controllers
γ	flow-setup delay bound
Decision variables	Descriptions
x_i^t	Binary variable indicating whether a controller is placed at switch
y_{ij}^t	Binary variable indicating whether the flow f_i is assigned to the controller c_j placed at s_j

However, the above work adopts a node-based assignment method, which can easily cause long waits for flow requests.

3 System Model and Problem Formulation

In order to realize the optimal placement and assignment of controllers, in this section, we model the problem as a multi-constraint optimization problem, and then show that it is a NP hard problem.

3.1 System Model

We consider an SDN-enabled edge network as depicted in Fig.2, which has multiple controllers to be deployed in the system. The SDN-enabled edge network is modeled as an undirected graph $G(V, E)$ with n nodes, where V represents the set of edge nodes, and E represents the set of network links between edge nodes. Let $S = \{s_1, s_2, ..., s_m\} \in V$ denote the switch set and $C = \{c_1, c_2, ..., c_k\}$ denote the controller set, where $m, k \in \mathbb{N}^+$. Additionally, the set of flows in the network is represented as $F = \{f_1, f_2, ..., f_n\}$ where n is a positive integer denoting the number of flows within a time slot. Assuming that the set of flows assigned to the controller c_j is denoted with C_j, we have $C_1 \cup C_2 \cup ... \cup C_k = F$.

To study the placement and assignment policy, We introduce two binary optimization variable x_i^t and y_{ij}^t. The variable x_i^t indicates whether a controller

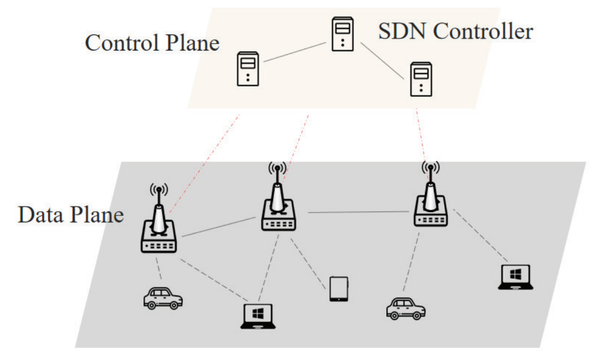


Fig. 2. An example of an SDN-enabled edge network.

is placed at switch s_i ($x_i^t = 1$) or not ($x_i^t = 0$). These variables constitute the controller placement policy as

$$P = (x_i \in \{0, 1\} : i = 0, 1, ..., n). \tag{1}$$

The other variable y_{ij}^t represents whether the flow f_i is assigned to the controller c_j placed at s_j ($y_{ij}^t = 1$) or not ($y_{ij}^t = 0$). These variables constitute the flow assignment policy as

$$A = (y_{ij} \in \{0, 1\} : i = 0, 1, ..., n, j = 0, 1, ..., m). \tag{2}$$

3.2 Flow-Setup Delay Model

When a switch receives a flow, it sends a Packet-in message to the controller, and the controller processes and replies to it. The time taken for this entire process is referred to as the flow-setup delay. The flow-setup delay we consider consists of three delays: propagation delay, queuing delay and processing delay.

The communication between the controller and the switch needs to propagate through the channel between each other. Therefore, the propagation delay depends on the path delay which includes the process to send and to receive the request. According to the shortest path algorithm [21], we can get the shortest path between the switch and the controller. We assume that the switch associated with the flow request f_i is s. Consequently, the propagation delay can be defined as follows

$$D_{ij}^{\text{prop}}(t) = 2 * \delta_{s,j}. \tag{3}$$

where $\delta_{s,j}$ denotes the propagation delay of the shortest path between the switch s and the controller placed at s_j.

When the controller receives a Packet-in message, it puts it in its own message processing queue. We need to account for the delay that the message experiences while waiting to be processed in the queue, known as queuing delay. We refer to the [20] for the modeling of queuing delay. The queuing delay depends on the request arrival rate and the execution rate of the controller, and approximate the flow-request arrival rate to a Poisson Process, which is mathematically denoted as $\alpha_j(t) = \sum_{i=0}^{k} y_{ij}^t$. Considering the M/M/1 queuing model,the queuing delay of flow request f_i processed on the controller placed at s_j can be defined as follows

$$D_{ij}^{que}(t) = \frac{1}{\mu_j - \alpha_j(t)}. \tag{4}$$

where μ_j denotes the execution rate of the controller placed at s_j.

The processing delay is determined by the time it takes for the controller to calculate the forwarding path for the flow. Let's assume that the number of CPU cycles required by the controller to calculate the forwarding path of the flow f_i is w_i. The processing delay can be defined as follows

$$D_{ij}^{proc}(t) = \frac{w_i}{\mu_j}. \tag{5}$$

When a packet arrives at a switch and the switch does not find a corresponding entry in its flow table, the switch will send a flow request to the controller to establish the forwarding path for that flow. The controller then accepts and processes the request and sends the calculated forwarding path back to the switch. Therefore, the control response delay of the controller placed at s_j for a flow f_i is calculated as

$$D_{ij} = D_{ij}^{prop}(t) + D_{ij}^{que}(t) + D_{ij}^{proc}(t) \tag{6}$$

So the flow-setup delay of the flow f_i is presented as follows

$$D_i = \sum_{j=0}^{m} y_{ij}^t x_j^t D_{ij}. \tag{7}$$

Consequently, the average flow-setup delay in time slot t can be quantified as follows

$$D = \frac{1}{n} \sum_{i=1}^{n} D_i. \tag{8}$$

3.3 Problem Formulation

Our objective is to minimize the average flow-setup delay in the network. Based on the flow-setup delay model above, We formulate the optimization problem as flows

$$\begin{aligned}
&\min_{\boldsymbol{P},\boldsymbol{A}} \quad D \\
&\text{s.t. } \forall i \in [0,m], x_i^t \in \{0,1\} && (9)\\
&\forall i \in [0,n], j \in [0,m], y_{ij}^t \in \{0,1\} && (10)\\
&\forall i \in [0,n], j \in [0,m], y_{ij}^t \leq x_j^t && (11)\\
&\forall i \in [0,n], \sum_{j=0}^{m} y_{ij}^t = 1 && (12)\\
&\sum_{i=0}^{m} x_i^t = k && (13)\\
&\forall j \in [0,m], \alpha_j^t < \mu_j && (14)\\
&\forall i \in [0,n], D_i < \gamma && (15)
\end{aligned}$$

where constraint (9)-(11) ensure that each switch can have at most one controller placed on it, and each flow is assigned to exactly one node where a controller is placed. Constraint (12) guarantees a flow is assigned to only one controller. Constraint (13) specifies the total number of controllers in the system. Constraint (14) ensures that the arrival rate of flow requests should be less than the processing rate of the controller. Lastly, constraint (15) takes into account the delay bound for a flow request.

It can be demonstrated that the above problem is a generalization of the well-studied p-median problem [22] which is NP-hard by setting the flow-setup delay bound and controller processing capacity to infinite and setting the queuing delay and the processing delay to zero. As a result, direct utilization of mathematical programming solvers such as CPLEX [23] can be computationally expensive. To address this, we propose an approximation algorithm as an alternative approach.

4 Controller Placement and Flow Assignment

The decisions regarding controller placement and flow allocation are interdependent in this problem. To address this interdependency, we propose a two-stage algorithm. In the first stage, we optimize controller placement by traffic segmentation. This helps in effectively distributing the controllers across the network. In the second stage, we combine hop-count and controller request queue length considerations to assign flows to controllers. This approach ensures that flows are assigned to controllers in a way that reduces both communication delay and queuing delay. By dividing the optimization process into these two stages, we can achieve an efficient and effective solution for both controller placement and flow assignment.

Algorithm 1 Algorithm for Controller Placement

```
Input: Network topology $G$, Flow set $F$, Number of controllers $k$
Output: Controller placement scheme $P$
1: $P \leftarrow \emptyset$
2: for $i = 1$ to $k$ do
3:     $a \leftarrow$ a random node of $G.nodes$
4:     Start breadth first search from $a$ to get a largest subgraph $g$, the sum of flows
       in the subgraph $\leq \frac{|F|}{k}$
5:     $G \leftarrow G - g$
6:     $a \leftarrow$node with the highest degree in $g$
7:     $P.add(a)$
8: end for
```

4.1 Controller Placement

The placement of controllers should consider the traffic distribution across the network. In our algorithm design 1, we utilize the traffic distribution to divide the network, employing a breadth-first traversal approach to partition the graph into traffic slices. Each traffic slice aims to have a total traffic load of nodes that is close to the average load of the controllers. The controller is then placed at the node with the highest degree in the traffic slice.

Although not all flow requests from switches in the traffic slice may be assigned to the controller placed within that specific slice during the final flow assignment, this approach leads to a more evenly distributed traffic slicing in the network. As a result, the traffic load around each controller's placement location becomes nearly balanced, reducing the likelihood of high queuing delays for flow requests.

Furthermore, this approach helps prevent excessive controller placement in one particular area. If too many controllers are placed in a concentrated area, some controllers may remain idle while others become heavily overloaded. This resource waste can be avoided by distributing controllers evenly across different areas, ensuring that flow request queues are not backlogged and controllers can respond promptly to incoming requests.

4.2 Flow Assignment

When considering the assignment of a flow request, the optimal choice is to select the controller with the lowest response delay. Controllers that are closer to the switch that sends the flow request have shorter propagation delays between them. Additionally, assigning flow requests to more idle controllers helps minimize the delays in the flow request queues.

Our assignment algorithm 2 is based on a greedy strategy. It operates by assigning flow requests in a layer-by-layer manner, starting from the nodes where the controllers are placed. Each switch's flow requests are assigned based on the number of hops between the switch and the controllers and request queue length of the controllers. Specifically, for each flow request originating from a switch,

Algorithm 2 Algorithm for Flow Assignment

```
Input: Network topology $G$, Controller placement scheme $P$
Output: Flow assignment Scheme
 1: $q \leftarrow P$
 2: $l = 0$
 3: for all $n \in G$ do
 4:     $n.S \leftarrow \emptyset$
 5: end for
 6: while $q \neq \emptyset$ do
 7:     for $i = 1$ to $q.size$ do
 8:         $v \leftarrow q.pop()$
 9:         if $|v.S| = 1$ then
10:             assign $v$'s all flows to the $c \in v.S$
11:         else
12:             for each flow $f$ of $v$ do
13:                 $c \leftarrow$ the most idle controller in $v.S$
14:                 assign $f$ to $c$
15:             end for
16:         end if
17:         for each neighbour $e$ of $v$ do
18:             if $e$ has not been visited then
19:                 $q.put(e)$
20:                 $e.S \leftarrow v.S$
21:                 $e.l \leftarrow l + 1$
22:             else
23:                 if $e.l = l + 1$ then
24:                     $e.S \leftarrow e.S \cup v.S$
25:                 end if
26:             end if
27:         end for
28:     end for
29:     $l \leftarrow l + 1$
30: end while
```

we choose the controller with the smallest number of hops from that switch. If there are multiple controllers with the same number of hops, we prioritize the most idle controller for assignment.

By following this approach, we aim to minimize the response delay by selecting controllers that are physically close to the requesting switches and have lower levels of workload. This strategy ensures efficient flow assignment and reduces the overall delay experienced by flow requests in the network.

5 Experiment and Performance Evaluation

We conduct Python-based simulations to evaluate the performance of the our scheme. The AttMpls network topology we used in the simulation is from the Internet topology Zoo [24]. Different parameters and their values are presented in

Table 2. SIMULATION PARAMETERS

Parameter	Value
Number of controllers	[2,3,4,5]
Number of flows	10000-30000
Network topology	AttMpls [24]
Controller exec.rate	50-150
Delay bound	50-200 ms

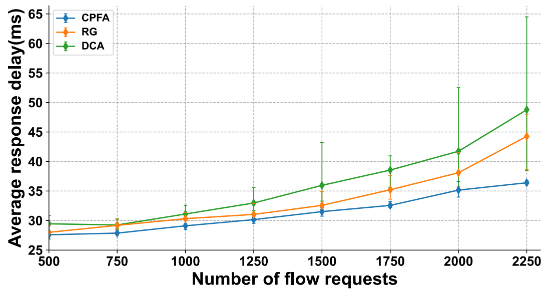


Fig. 3. Average controller response time with different number of flows.

Table 2. We compare our proposed scheme (CPFA) with two existing schemes: the Random Greedy algorithm (RG) [4] and Dynamic Controller Assignment (DCA) [20]. In the rest of the paper, we refer to CPFA, RG, and DCA to represent the proposed scheme, random greedy algorithm, and dynamic controller assignment scheme, respectively. The RG algorithm calculates the probability of placing controllers on nodes based on the optimization objective function, and obtains the corresponding controller-node assignment scheme according to the calculated set of controllers. On the other hand, DCA assigns flows to controllers according to preference lists. In contrast, our proposed scheme(CPFA) takes into account the influence of controller placement, effectively utilizing traffic distribution to partition traffic in the network. It also adopts a more fine-grained flow assignment method based on the greedy strategy. To evaluate the performance, we use the following metrics: average control response delay, maximum control response delay, the percentage of flows with high queuing delay, and the percentage of flows that violate the delay constraint. These metrics demonstrate the effectiveness and superiority of our proposed scheme. The experimental results are discussed in the following subsections.

5.1 Controller Response Time

We measure the average controller response delay and the maximum controller response delay for different numbers of flows and controllers. As shown in the Fig.3-5, our proposed scheme outperforms the existing schemes (RG and DCA)

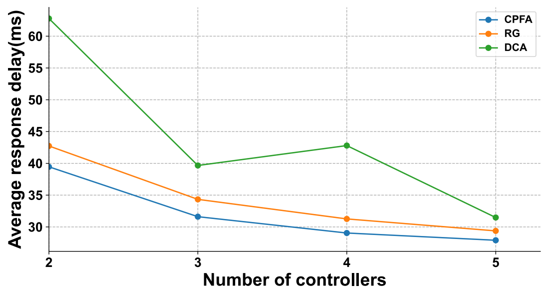


Fig. 4. Average controller response time with different number of controllers.

in terms of controller response delay. When the number of flows in the network is 2250, our scheme achieves a 34.0% improvement over DCA and a 21.6% improvement over RG in terms of average flow-setup delay.

The superiority of our scheme can be attributed to two key factors. Firstly, our scheme leverages the traffic distribution in the network to determine the optimized controller locations. This enables more efficient distribution of flow requests across controllers. In contrast, the DCA scheme does not consider the impact of controller location on system performance. As a result, it may lead to a scenario where only a few controllers handle a high traffic load, while other controllers remain idle. This imbalance reduces the overall working performance of the control plane, with overloaded controllers experiencing increased response delay and idle controllers wasting resources. Secondly, our scheme optimizes flow request assignment based on delay requirements and controller workload. This approach reduces the likelihood of response time timeouts caused by controller overload and ensures that flow requests are assigned in a way that helps minimize queuing delay. In contrast, the existing RG scheme assigns edge nodes to controllers, and each controller handles all the flow requests from the assigned nodes within a given period. If a controller receives a large number of flow requests from managed nodes in a short period of time, the queuing delay of flow requests can significantly increase, resulting in high controller response delay.

Furthermore, it is evident that the controller response delay increases with an increase in the number of flows and decreases with an increase in the number of controllers from Fig.3-5. This relationship can be attributed to the fact that a higher number of flows results in a single controller handling a larger volume of flow requests, leading to increased queuing delay and, consequently, higher controller response delay. On the other hand, deploying a larger number of controllers in the network reduces the average number of flow requests that each controller needs to handle. This allows for more efficient assignment of flow requests to controllers with lower propagation delay and higher levels of idleness within the network. As a result, the controller response delay is reduced. Surprisingly, when the number of controllers changes from 3 to 4, the average response delay of DCA increased as shown in 4. We investigate the reason and find that

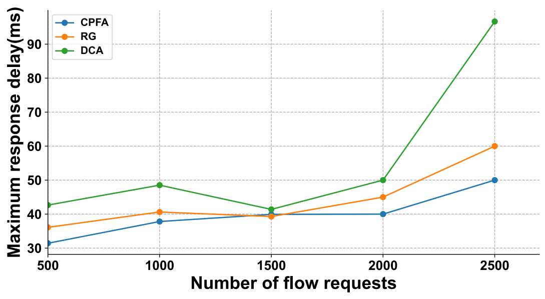


Fig. 5. Maximum controller response time with different number of flows.

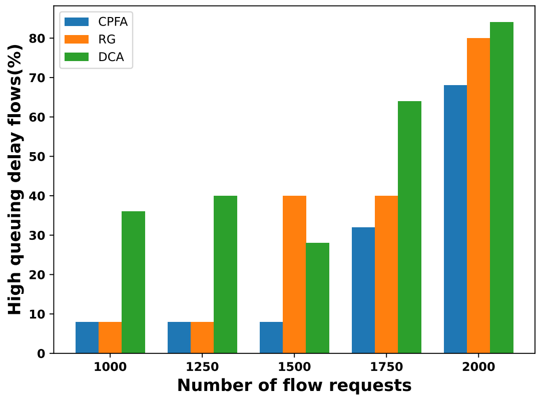


Fig. 6. Percentage of high queueing delay flows with different number of flows.

it is due to the DCA algorithm randomly generating a terrible controller plane layout when the number of controllers is 4.

5.2 High Queuing Delay Flows

We also conduct an experiment to measure the percentage of flows experiencing high queuing delay, and the results demonstrated that our scheme outperforms RG and DCA. RG is a static assignment scheme that assigns all flow requests from an edge node to its designated controller. However, when the number of flow requests from an edge node increases significantly within a time slot, it leads to a backlog of flow requests in the controller's request processing queue, resulting in high queuing delays. On the other hand, DCA does not take into account the placement of controllers in the network. It is influenced by the convergence of transmission delay and queuing delay while maintaining the preference list of flows. This may result in higher priority in the priority queue of controllers with

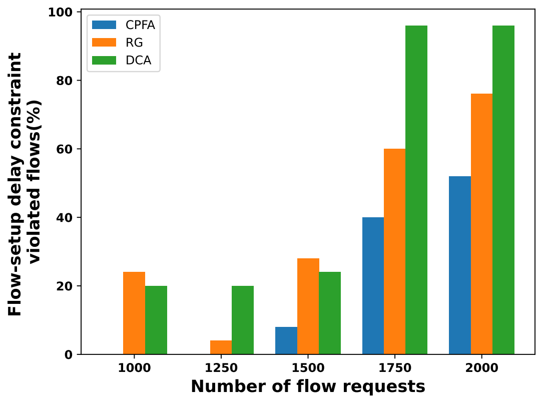


Fig. 7. Percentage of delay constraint violated flows with different number of flows.

relatively high queuing delay due to low propagation delay, leading to increased delays in request queuing.

In contrast, our scheme is based on dynamic flow assignment. After optimizing controller placement, for controllers with close transmission delays, we prioritize selecting relatively idle controllers to process requests. This approach reduces the queuing time of flow requests and enhances the efficiency of the controller plane.

5.3 Folw-setup Delay Constraint Violated Flows

We also measure the percentage of flows in the network that violated the flow-setup delay constraint. The results clearly demonstrate that our proposed scheme outperforms the existing schemes RG and DCA in this aspect. Our proposed scheme takes the constraints of both the controller response delay and the flow delay bound into consideration. By incorporating these constraints, we are able to ensure that a larger number of flows receive responses with low delays, within the specified constraints. Consequently, only a small number of flows exceed the setup delay bound.

In contrast, the RG scheme does not consider the delay bound, and its node-based assignment method is more prone to overlooking the controller response delay of individual flows. As a result, some flow requests end up queuing for extended periods in the request queues of certain controllers, leading to high controller response delays for those flows. Consequently, these flows are more likely to violate the flow-setup delay bound. Although DCA takes the flow-setup delay bound into account, it lacks a decision-making process for controller placement. As a result, random placement of controllers can potentially lead to an inefficient layout of the controller plane in the network. This can result in a significant number of flow requests requiring relatively high propagation delays

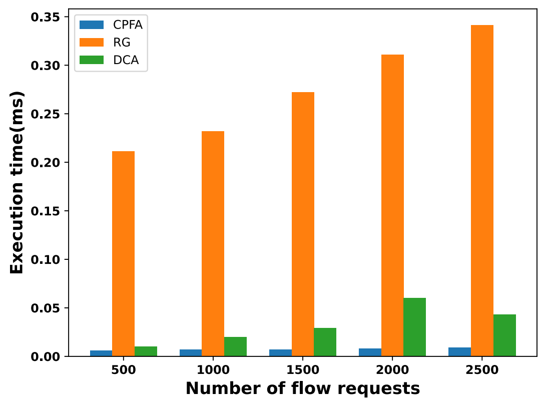


Fig. 8. Algorithm execution time with different number of flows.

during processing, leading to a substantial increase in the percentage of flows that violate the flow-setup delay bound.

Furthermore, it is worth noting that the percentage of flows violating the flow-setup delay threshold increases with the number of flows, while the RG scheme decreases at 1250 due to the randomness of the algorithm. This is because as the number of flows increases, the controller response delay also tends to increase. Consequently, more flows exceed the specified delay bound.

5.4 Algorithm Execution Time

We also compare the execution time of the algorithms, and the results indicate that our algorithm achieves a more efficient execution process by optimizing the algorithm design and considering the specific requirements of the problem. This observation is also supported by the theoretical analysis of the algorithm. Our algorithm is based on traversal and greedy strategies, which has a linear complexity. In contrast, DCA involves calculating preference lists for controllers and performing bidirectional matching for flow requests, resulting in higher computational complexity. The number of iterations of the RG algorithm depends on the number of nodes that can play the role host for a controller, and in each iteration, the optimal assignment strategy under the current placement strategy needs to be calculated, resulting in relatively high algorithm complexity.

6 Conclusion

To enhance the programmability and scalability of edge networks, multiple controllers need to be placed in the network to effectively manage the network traffic in the data plane. Well-placed controllers in the network and fine-grained assignment of flows to controllers can effectively shorten flow-setup delay in the network, thereby improving network performance. In this paper, we consider the problem of controller placement and flow assignment in edge networks. To address the issue of high controller response delay caused by excessively long request queues of some controllers, we propose a controller placement and fine-grained flow assignment scheme based on traffic segmentation, with the objective of minimizing the average flow setup delay in the network. The evaluation results show that compared with existing solutions, our proposed scheme has better performance in controller response time, percentage of high queuing delay flows, percentage of flow-setup delay constraint violated flows, and algorithm execution time.

References

1. Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.
2. Yue Zeng, Songtao Guo, and Guiyan Liu. Comprehensive link sharing avoidance and switch aggregation for software-defined data center networks. *Future Generation Computer Systems*, 91:25–36, 2019.
3. Pan Li, Guiyan Liu, Songtao Guo, and Yue Zeng. Traffic-aware efficient consistency update in NFV-enabled software defined networking. *Computer Networks*, 228:109755, 2023.
4. Qiaofeng Qin, Konstantinos Poularakis, George Iosifidis, and Leandros Tassiulas. Sdn controller placement at the edge: Optimizing delay and overheads. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 684–692. IEEE, 2018.
5. Yue Zeng, Songtao Guo, Guiyan Liu, Pan Li, and Yuanyuan Yang. Energy-efficient device activation, rule installation and data transmission in software defined DCNs. *IEEE Transactions on Cloud Computing*, 10(1):396–410, 2019.
6. Yue Zeng, Baoliu Ye, Bin Tang, Sanglu Lu, Feng Xu, Song Guo, and Zhihao Qu. Mobility-Aware Proactive Flow Setup in Software-Defined Mobile Edge Networks. *IEEE Transactions on Communications*, 71(3):1549–1563, 2023.
7. Pan Li, Songtao Guo, Chengsheng Pan, Li Yang, Guiyan Liu, and Yue Zeng. Fast congestion-free consistent flow forwarding rules update in software defined networking. *Future Generation Computer Systems*, 97:743–754, 2019.
8. Penghao Sun, Zehua Guo, Gang Wang, Julong Lan, and Yuxiang Hu. Marvel: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning. *Computer Networks*, 177:107230, 2020.
9. Xi Huang, Simeng Bian, Ziyu Shao, and Hong Xu. Predictive switch-controller association and control devolution for sdn systems. *IEEE/ACM Transactions on Networking*, 28(6):2783–2796, 2020.
10. Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. *ACM SIGCOMM Computer Communication Review*, 42(4):473–478, 2012.
11. Zehua Guo, Weikun Chen, Ya-Feng Liu, Yang Xu, and Zhi-Li Zhang. Joint switch upgrade and controller deployment in hybrid software-defined networks. *IEEE Journal on Selected Areas in Communications*, 37(5):1012–1028, 2019.
12. Yiwen Wu, Sipei Zhou, Yunkai Wei, and Supeng Leng. Deep reinforcement learning for controller placement in software defined network. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1254–1259. IEEE, 2020.

13. Deborsi Basu, Abhishek Jain, Uttam Ghosh, and Raja Datta. A reverse path-flow mechanism for latency aware controller placement in vsdn enabled 5g network. *IEEE Transactions on Industrial Informatics*, 17(10):6885–6893, 2020.
14. EL Hocine Bouzidi, Abdelkader Outtagarts, Rami Langar, and Raouf Boutaba. Dynamic clustering of software defined network switches and controller placement using deep reinforcement learning. *Computer Networks*, 207:108852, 2022.
15. Long Chen, Feilong Tang, and Xu Li. Mobility-and load-adaptive controller placement and assignment in leo satellite networks. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.
16. Bo Li, Xiaoheng Deng, and Yiqin Deng. Mobile-edge computing-based delay minimization controller placement in sdn-iov. *Computer Networks*, 193:108049, 2021.
17. Yuepeng Li, Deze Zeng, Lvhao Chen, Lin Gu, Weiyin Ma, and Feng Gao. Cost efficient service mesh controller placement for edge native computing. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 1368–1372. IEEE, 2022.
18. Reza Soleymanifar, Amber Srivastava Carolyn Beck, and Srinivasa Salapaka. A clustering approach to edge controller placement in software-defined networks with cost balancing. *IFAC-PapersOnLine*, 53(2):2642–2647, 2020.
19. Junjie Xie, Deke Guo, Xiaozhou Li, Yulong Shen, and Xiaohong Jiang. Cutting long-tail latency of routing response in software defined networks. *IEEE Journal on Selected Areas in Communications*, 36(3):384–396, 2018.
20. Samaresh Bera, Sudip Misra, and Niloy Saha. Traffic-aware dynamic controller assignment in sdn. *IEEE Transactions on Communications*, 68(7):4375–4382, 2020.
21. Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
22. Mark S Daskin and Kayse Lee Maass. The p-median problem. In *Location science*, pages 21–45. Springer, 2015.
23. IBM ILOG. Cplex optimizer. *En ligne]. Available: http://www-01. ibm. com/software/commerce/optimization/cplex-optimizer*, 2012.
24. Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.