scientific reports

Check for updates

OPEN

QOGMP: QoS-oriented global multi-path traffic scheduling algorithm in software defined network

Yiping Guo[1]✉, Guyu Hu[1,3] & Dongsheng Shao[2,3]

According to the research status of Software Defined Network (SDN) control layer traffic scheduling, we find the current common problems, including single path, easy congestion, Quality of Service (QoS) requirements and high delay. To solve these four problems, we design and implement a QoS-oriented global multi-path traffic scheduling algorithm for SDN, referred to as QOGMP. First, we propose a link weight calculation algorithm based on the idea of traction links and deep reinforcement learning, and conduct experimental verifications related to traction links. The algorithm considers QoS requirements and alleviates the problems of easy congestion and high delay. Then, we propose a traffic scheduling algorithm based on link weight and multi-path scheme, which also considers QoS requirements and solves the problem of single path. Finally, we combined the link weight calculation algorithm and the traffic scheduling algorithm to implement QOGMP, and carried out comparative experiments in the built simulation environment. The experimental results show that QOGMP is better than the two comparison algorithms in terms of delay and rescheduling rate.

With the continuous expansion of network scale and continuous updating of network technology, high availability and high performance requirements are put forward to the network. In order to maintain network availability and improve network performance, it is necessary to effectively allocate network resources and reasonably schedule network traffic[1]. Traffic scheduling is the process of assigning concurrent request packets of massive users to server program instances of different IP addresses according to a specific strategy.

With the continuous increase in the number and types of devices connected to the network, in the face of the rapidly changing business environment of the Internet, the drawbacks of the traditional network have gradually become apparent[2]. Under the new situation of rapid network development and continuous emergence of network applications, traffic scheduling based on the traditional network has limitations: (1) It challenges the efficiency of traffic scheduling. The development of cloud computing has made the demand for large-scale data centers more and more obvious. In the process of network traffic scheduling, network resources should be used more effectively to reduce costs and improve the overall performance of the network[3]. However, the traditional network traffic scheduling technology has problems such as low performance and high overhead when facing large-scale networks. (2) The network is prone to congestion. In traditional network traffic scheduling, only local information can be used. The network node calculates the path to the destination node based on itself. It does not have a global network view, so it is difficult to perform good traffic scheduling[4], which makes the probability of congestion and other problems in the network continue to increase. (3) Traffic scheduling is not flexible. In the face of rapid network changes and user demand, it is very weak. After traffic scheduling according to network application requirements, if the demand changes, the configuration of the corresponding network equipment (such as routers, switches and firewalls) needs to be revised in the traditional network[5], so as to re-schedule the traffic, which is a very tedious process.

The birth of Software Defined Network (SDN) can solve these limitations. In 2009, Professor Mckeown formally proposed SDN, a new network architecture in the literature[6], breaking the closed mode of integration of software and hardware of traditional network equipment, and separating the control level of network equipment from the data forwarding level.

[1]Command and Control Engineering College, People's Liberation Army Engineering University, Nanjing, CO 210007, China. [2]Unit 31106 of People's Liberation Army, Nanjing, CO 210007, China. [3]These authors contributed equally: Guyu Hu and Dongsheng Shao. ✉email: 2609529342@qq.com

The design concept of SDN is to separate the control layer and data layer of the network while realizing programmable control, which can provide centralized management and dynamic maintenance capabilities for distributed networks[7,8], thereby effectively solving the disadvantages of the traditional IP network in maintenance, expansion, and experimental innovation. The typical architecture of SDN is divided into three layers[9]. The top layer is the application layer, including various services and applications. The middle layer is the control layer, which is mainly responsible for processing the data resource arrangement and maintaining information such as network topology and status. The main body of the control layer is a logically centralized and programmable controller that can master global network information, which is convenient for operators and scientific researchers to manage and configure the network and deploy new protocols. The bottom layer is the data layer, which is responsible for data processing, forwarding, and status collection based on the flow table. The main body of the data layer is a lot of dumb switches (different from the traditional two-layer switches, specifically refers to the equipment used to forward data). These switches only provide simple data forwarding functions and can quickly process matched data packets to meet the increasing demand of traffic. An open unified interface (such as OpenFlow[10]) is used to interact between the control layer and the data layer. The controller issues unified standard rules to the switch through the standard interface, and the switch only needs to perform corresponding actions in accordance with standard rules.

Different from the "slice" management of the traditional network[11], the control layer can use the global network view and dynamic rule configuration capabilities provided by SDN to perform load balancing and flexible traffic scheduling[12]. This solves the limitations of traffic scheduling based on the traditional network to a large extent, thereby maintaining network availability and improving network performance. Therefore, it is of great significance to carry out research on SDN-based control layer traffic scheduling methods.

Research status and content

Research status. In recent years, many scholars have devoted themselves to the study of SDN-based control-layer traffic scheduling methods. These researches are dedicated to solving different problems, including four problems such as single path, easy congestion, QoS requirements, and high latency.

(1) Single path. The means to solve the single-path problem in literature[13–17] are all multi-path transmission. For example, literature[13] proposed an equal-cost multi-path (ECMP) scheme, which is currently widely used. However, this type of solution has two major problems: one is that the multi-path transmission scheme implemented under specific conditions lacks versatility in the SDN environment; the other is that traffic scheduling can only use local information, which is likely to cause congestion problems.

(2) Easy to congest. The traffic in the network has shown explosive growth. The traditional network architecture cannot achieve flexible, fast and effective scheduling of network traffic. In addition, it is difficult to know the load status of the path. Congestion problems are prone to occur, resulting in low link utilization. The means to solve the problem of easy congestion in literature[18–23] include processing elephant flows and using SDN global link load information. These methods introduce additional overheads such as query detection and congestion calculation, which increase the forwarding delay to a certain extent.

(3) QoS requirements. When the network is overloaded or congested, QoS can ensure that important services are not affected by delay or packet loss during the transmission process[24], while ensuring the efficient operation of the network. At present, SDN can provide QoS guarantee through mechanisms such as flow control and bandwidth reservation, but it is difficult to meet the increasing demand for QoS of business applications[25]. A real QoS-oriented traffic scheduling scheme is needed. The main means to solve the QoS problem in literature[26–29] is to introduce user-defined constraints and comprehensively consider the link occupancy rate and the size of the business flow. These methods also introduce additional storage overhead such as packet loss rate measurement and link occupancy calculation and the delay caused by this. There is no widely used standard for user-defined constraints, and even the SDN northbound interface (between the application layer and the control layer) as the basis for its realization has not yet a unified or recognized standard. Although its research space is relatively large, its current practicability and research value are not great.

(4) High latency. The solutions to the above three problems will introduce considerable delay pressure. In addition, in the face of large-scale networks, the current routing algorithms are obviously weak. It is necessary to consider an efficient forwarding path calculation algorithm to deal with it. Machine learning algorithms can usually extract traffic characteristics automatically, and do not rely on expert experience, so they are more efficient than traditional solutions in solving traffic scheduling problems. Literature[30] proposed an SDN global multi-path traffic scheduling algorithm based on reinforcement learning. The algorithm uses the link bandwidth information provided by the SDN data layer to update the link weights, and selects k shortest paths as the forwarding paths according to the weights. Such methods based on reinforcement learning are difficult to classify complex traffic characteristics. Literature[31] proposed an SDN traffic scheduling algorithm based on the deep neural network. Such methods are based on deep learning. Deep learning can make up for the shortcomings of reinforcement learning but requires a large amount of labeled data to train neural networks.

Research content. Based on the above research status, in view of the four existing problems in the SDN control layer traffic scheduling research, we consider the following solutions: (1) Using the multi-path scheduling method of the ECMP scheme to solve the single path problem. (2) Using the global information provided by the centralized controller of SDN to alleviate the congestion problem. (3) Calculating traffic forwarding rules by integrating multiple link parameters (packet loss, delay, link capacity) and the size of business traffic to provide

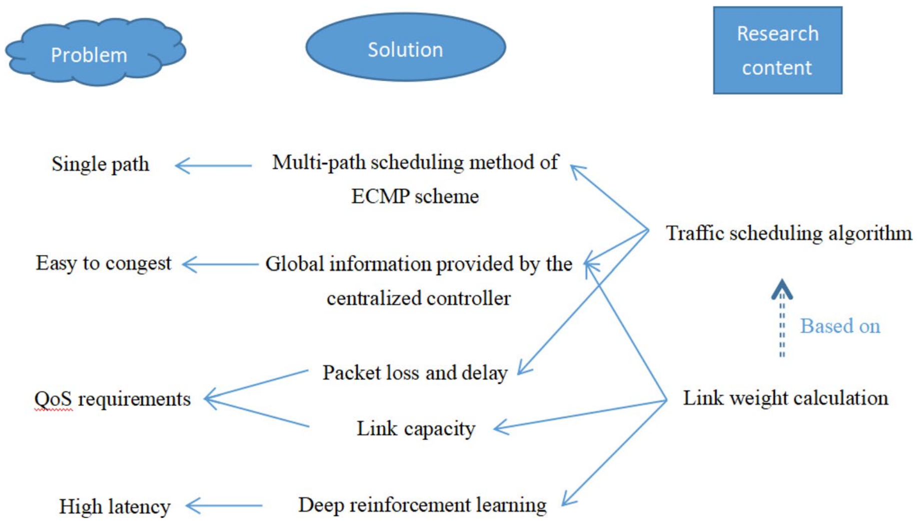
Problem
Solution
Research content
Single path
Multi-path scheduling method of ECMP scheme
Easy to congest
Global information provided by the centralized controller
Traffic scheduling algorithm
Based on
QoS requirements
Packet loss and delay
Link capacity
Link weight calculation
High latency
Deep reinforcement learning

Figure 1. Main research content.

QoS guarantee. (4) Using deep reinforcement learning algorithms to alleviate the high latency problem. The forwarding path calculation problem is suitable for deep reinforcement learning, and deep reinforcement learning algorithms can overcome the deficiencies of reinforcement learning and deep learning.

QoS refers to the service capability that a network can provide for network communication tasks. For different communication tasks, QoS needs to achieve different indicators, including packet loss tolerance, delay tolerance, link capacity and other metrics[32]. In the traditional single-path transmission mode, all QoS-related network parameters can be used for link weight calculation and the optimal path can be selected. However, when we adopt multi-path transmission, we need to balance multiple paths, considering the efficiency of the path selection scheme and the success rate of scheduling. Therefore, in order to ensure the efficiency of the path selection scheme, we combine the deep reinforcement learning algorithm to use the packet loss and delay parameters for link weight calculation. In order to improve the scheduling success rate of the scheme, we use the link capacity parameter to calculate the traffic forwarding path.

The main research content is as shown in Fig. 1. The paper structure is as follows: (1) In section two, we propose a link weight calculation algorithm based on the idea of traction link and deep reinforcement learning, and conduct related experiments to verify the effectiveness of traction link. This algorithm provides QoS guarantee and alleviates the problems of easy congestion and high delay. (2) In section three, we propose a traffic scheduling algorithm based on link weight and multi-path scheme, which also considers QoS requirements and solves the problem of single path. (3) In section four, we combine the algorithms proposed in section two and section three to implement a QoS-oriented global multi-path traffic scheduling algorithm for SDN, or QOGMP for short. We conduct comparative experiments in the built simulation test environment. The experimental results show that the performance of QOGMP is better than that of the algorithms for comparison.

Link weight calculation algorithm

We design a link weight calculation algorithm based on the idea of traction link and deep reinforcement learning.

We use deep reinforcement learning algorithms to calculate link weights. The agent of the reinforcement learning system is set as a neural network (strategy generation network), and its interaction with the environment is modeled as a Markov process. The Markov process is represented by a four-tuple $E =< X, A, P, R >$, where the probability P defaults to 1, the state of the environment $x \in X$ is the current traffic view, the action $a \in A$ is the link weight value, and the reward $r \in R$ is the strategic value feedback provided by the environment for the neural network. By constantly trying actions, the strategy π is obtained, and the action to be executed $a = \pi(x)$ can be known in the state x. The quality of the strategy can be measured by the value function.

The strategy generation network is implemented by the neural network $a = \pi(x|\mu)$, and its parameter is μ. The strategic value network is realized by the neural network $Q(x_t, a_t|\theta)$, and its parameter is θ.

The value function of the strategy is expressed as Eq. (1).

$$Q(x_t, a_t) = r_t + \gamma Q(x_{t+1}, a_{t+1}). \tag{1}$$

The reward function of the strategy is expressed as Eq. (2).

$$y_t = r_t + \gamma Q(x_t, \pi(x_{t+1})|\theta). \tag{2}$$

The parameters of the strategy value network and the strategy generation network need to be continuously adjusted based on the error of value and reward.

The parameter update of the strategic value network is in accordance with Eq. (3).

$$\nabla_\theta = E[(Q(x_t, a_t|\theta) - y_t)^2]. \tag{3}$$

The parameter update of the strategy generation network is in accordance with Eq. (4).

$$\begin{aligned}\nabla_\mu &\approx E[\nabla_\theta Q(x, \pi(x|\mu)|\theta)]\\ &= E[\nabla_a Q(x, a|\theta)\nabla_\mu \pi(x|\mu)].\end{aligned} \tag{4}$$

The input of the strategy generation network is the traffic view and reward, and the output is the link weight value we need. The traffic view is a summary of the link information collected and calculated by the data layer, including information such as nodes, links, and the cost of each link. Since different QoS requirements have different requirements for packet loss and delay, we express the link cost as Eq. (5).

$$cost = \alpha \cdot packetloss + \beta \cdot delay. \tag{5}$$

In Eq. (5), α and β are coefficients set according to user needs($\alpha + \beta = 1$). In the following experiments, α and β are both set to 0.5.

The flow of link weight calculation algorithm based on traction link and deep reinforcement learning is shown in Algoritnm 1.

Algorithm 1 link weight calculation

```
1:  Input: Traffic view $x_1$
2:  Initialize parameters μ and θ
3:  for $i = 1$ to $M$ do
4:      Initialize Noise//Add random noise to the action to improve the strategy exploration effect
5:      for $t = 1$ to step do
6:          $a_t = \pi(x_t|\mu) +$ Noise
7:          Receive $r_t$ and $x_{t+1}$ from the environment
8:          Update the parameter   θ  according to Eq. (3)
9:          Update the parameter   μ  according to Eq. (4)
10:     end for
11: end for
12: Output: Link weight $a$
```

Assuming that the number of network nodes is n, the maximum number of optional links increases exponentially, as shown in Eq. (6).

$$C_{n-2}^0 + C_{n-2}^1 + \cdots + C_{n-2}^{n-2} = 2^{n-2}. \tag{6}$$

Although deep reinforcement learning algorithms have strong computing power, in the face of such a huge amount of data, we still need to consider reducing the number of optional links. We adopt the idea of traction link proposed in literature[33] to alleviate this problem. Traction control theory points out that for the control of large-scale networks, it is only necessary to apply control signals to some nodes, and realize the diffusion of control signals through the connection relationship between nodes, and finally realize the coordination of the whole network, so as to achieve the control goal. For example, if there are 100 original links, the original method is to update the weight of 100 links, and finally get the path scheme L. Now, we extract 20 traction links from the original links, and update the weights of these 20 links. According to the traction control theory, the final path scheme is still L with great probability.

In the weight calculation phase, we replace original links with traction links included in original links to achieve the goal of not affecting the path selection result after the weight update, but greatly reducing the number of links to be processed.

Taking the link graph collected by the data layer as input, the flow of the traction link extraction algorithm is shown in Algoritnm 2.

Algorithm 2 Traction link extraction

```
1:  Input: link diagram $G = (V, E)$
2:  Initialize the searched node set $V_{done} = \varnothing$
3:  Initialize the traction link set $M = \varnothing$
4:  Select the node with the smallest degree in $V$ and add it to the set $V_{doing}$, add other nodes to the set $V_{undone}$
5:  Flag=0
6:  while $V_{done} \neq V$ do
7:      for all nodes $v_i \in V_{doing}$ do
8:          for all nodes $v_j \in neighbour(v_i)$ do
9:              if $v_i \in V_{undone}$ then
10:                 $Flag \div 2 = a \cdots\cdots g$
11:                 if g=0 then
12:                     Add $e(v_i, v_j) \in E$ to $M$
13:                     Move $v_j$ out of $V_{undone}$ and add to $V_{doing}$
14:                 end if
15:             end if
16:         end for
17:         Flag=Flag+1
18:         Move $v_i$ out of $V_{doing}$ and add to $V_{done}$
19:     end for
20: end while
21: Output: Traction link diagram $T = (V, M)$
```

We use the traction link graph output by Algorithm 2 as the input of Algorithm 1.

Traffic scheduling algorithm

We design a global multi-path traffic scheduling algorithm based on link weight and ECMP. Our ultimate goal is to generate a traffic scheduling scheme, that is, to calculate the traffic forwarding path and the traffic distribution on each path.

Different from the single-path scheme, the multi-path scheme may have uneven traffic distribution, resulting in low link utilization or even close to congestion, resulting in high delay and low network throughput, so that the scheduling scheme is unsuccessful and enters rescheduling. Therefore, after we update the link weight, we solve this problem by balancing the link capacity and service flow of multiple alternative paths. The link capacity and service flow data are collected by the data layer and fed back to the control layer.

How many paths do we need for traffic matching? Is it all paths? Of course not, this answer needs to be studied on the ECMP scheme to get it. The ECMP scheme is a general multi-path traffic scheduling scheme at present, which can map a single flow to multiple paths. In general, if a flow is mapped to too many paths, the delay is low but the link utilization is too low. Conversely, if a flow is mapped to too few paths, the link utilization is high but the delay is too high. We need to explore how many links a flow can be mapped to achieve the optimal compromise between link utilization and delay. To this end, we did a simple experiment.

We assume that in a network environment, the service flow is fixed at 1 Mb and evenly distributed, and the delay is only the transmission delay. All links between the source address and the destination address are black boxes, and only the transmission rate and number of the link are known. We define a standard for the compromise between link occupancy and delay. The value of the performance compromise is equal to the product of each link occupancy divided by the longest delay in all links. The larger the value, the better the result. The definition of each link occupancy is the link delay divided by the longest delay in all links. The definition of the link delay is the service traffic allocated to the link divided by the transmission rate of the link.

When the number of links is 3, the performance compromise is calculated for different link mapping schemes and transmission rates, and the results are shown in Table 1. The number of mapped links is n, that is, the first n links are selected to calculate the performance compromise.

It can be drawn from Table 1 that the number of mapped links with the best performance compromise is 2 when the number of links is 3.

The above experimental process is a calculation from the original number of links to the number of mapped links with the best performance compromise. We perform similar calculations and records for different numbers of links, and summarize the number of mapped links with the best performance compromise, as shown in Table 2.

It can be obtained from Table 2 that the number of mapped links with the best performance compromise is $[\sqrt{n}]$ when the number of mapped links is n. Therefore, we take $[\sqrt{n}]$ paths for traffic distribution.

There are two traffic ratio schemes: the first is to configure the ratio according to the weight on the premise that the link capacity reaches a certain requirement (to ensure that the delay is acceptable). The second is to simply allocate according to the margin ratio. The definition of the margin ratio is the capacity of each path to be allocated divided by the total capacity of all paths to be allocated. The capacity of each path is the minimum value of the capacity of all links on the path.

The number of mapped links	Transmission rate of link 1, 2, 3 (Mb/s)	Performance compromise
1	1, 2, 3	1
2	1, 2, 3	1
3	1, 2, 3	0.5
1	2, 3, 4	1
2	2, 3, 4	2.67
3	2, 3, 4	2
1	3, 4, 5	1
2	3, 4, 5	4.5
3	3, 4, 5	4.05

Table 1. The Performance compromise record when the number of links is 3.

n	b	n	b	n	b	n	b	n	b	n	b	n	b	n	b	n	b	n	b
3	2	4	2	5	2	6	2	7	3	8	3	9	3	10	3	11	3	12	3
14	4	16	4	18	4	20	4	22	5	24	5	26	5	28	5	30	5	32	6
35	6	38	6	41	6	44	7	47	6	50	7	53	7	56	8	59	8	62	8

Table 2. This is a record table, where n is the number of mapped links and b is the number of mapped links with the best performance compromise.

Algorithm 3 QOGMP

```
1: Input: (1)$[\sqrt{n}]$ weighted shortest paths; (2) business traffic size $S$; (3) delay tolerance $D$; (4) the weight of each path to be allocated $w_1, w_2, \cdots, w_{[\sqrt{n}]}$; (5) the capacity of each path to be allocated $c_1, c_2, \cdots, c_{[\sqrt{n}]}$
2: the traffic of each path to be allocated is $S_i = S \cdot w_i/(w_1, w_2, \cdots, w_{[\sqrt{n}]})$
3: for $i = 1$to$[\sqrt{n}]$ do
4:     if $S_i/c_i > D$ then
5:         break
6:     end if
7: end for
8: if $i == [\sqrt{n}]$ then
9:     print $S_1, S_2, \cdots, S_{[\sqrt{n}]}$
10: else
11:     $S_i = S \cdot c_i/(w_1, c_2, \cdots, c_{[\sqrt{n}]})$
12:     for $j = 1$ to $[\sqrt{n}]$ do
13:         if $S_i/c_i > D$ then
14:             break
15:         end if
16:     end for
17:     if $j == [\sqrt{n}]$ then
18:         print $S_1, S_2, \cdots, S_{[\sqrt{n}]}$
19:     else
20:         reschedule
21:     end if
22: end if
23: Output: $[\sqrt{n}]$paths and their traffic distribution
```

We update the link weight according to the output of Algorithm 1. If the weight is not updated, the default is 1. After calculating the $[\sqrt{n}]$ weighted shortest paths iteratively using Dijkstra's algorithm (execute Dijkstra algorithm to find the first shortest path from the source point to the destination point, remove the first one and execute the algorithm again to find the second shortest path, and iterate until $[\sqrt{n}]$ paths are found), we execute Algorithm 3 to generate a traffic scheduling scheme.

Simulation implementation and experiment

Simulation implementation.

Pycharm can not only run python algorithms, but also create graphical interfaces. We use the editor pycharm-community-2019.1.1 to implement the algorithm and create a simulation environment.

Data set	The number of original links	The number of traction links
1	4	2
2	11	3
3	20	5
4	20	5
5	31	7
6	40	11
7	54	15
8	56	19
9	57	18
10	60	22
11	62	22
12	66	22
13	67	27
14	70	26
15	80	29
16	92	34
17	92	33
18	97	38
19	99	37
20	100	37

Table 3. The extraction result of traction link.

We combine the algorithms in section two and section three to implement the complete flow scheduling algorithm QOGMP, which is implemented in the order of Algorithm 2, Algorithm 1, and Algorithm 3. We perform a simplified simulation of the SDN network system.

The simplified network system is divided into two layers: the control layer and the data layer. The controller of the control layer has the function of receiving information from the data layer and the calculation function of the traffic scheduling scheme. The data layer has the function of data collection and flow forwarding. Data transmission is allowed between the two layers.

In the order of execution, the specific functional design ideas are as follows: (1) Data collection function of the data layer: The data layer collects network information, including switch V, link E and link parameters (cost C, link capacity W). (2) The function of the controller to receive information from the data layer: The controller obtains the traffic view $G(V, E, C, W)$ fed back from the data layer. (3) The function of the controller to calculate the flow scheduling scheme: We embed the traffic scheduling algorithm into the controller as the main algorithm of the controller. We take the traffic views $G1(V, E, C)$, $G2(V, E, W)$ and user requirements (including business flow and delay tolerance) as input. After the main algorithm is executed, the traffic scheduling scheme is output. (4) Flow forwarding function of the data layer: The data layer receives the scheme generated by the controller and forwards the flow according to it (the stage task is to calculate the transmission delay).

Verification experiment of traction link.

We conduct related experiments to verify the effectiveness of traction link. After implementing Algorithm 2, we record the amount of calculation saved after applying the traction link algorithm to verify whether the extraction of the traction link can greatly reduce the number of links.

We conduct experiments on 20 link graphs $G = (V, E)$ with different link numbers. Assuming that there are n nodes in V, the input format of the link graph (that is, the content of the data set) is a $n \times n$ numerical matrix, $e(i, j) = 1$ indicates that there is a link between node i and node j, $e(i, j) = 0$ means that there is no link between node i and node j. The data used in the experiment is randomly generated. We record the number of links in the input and output link graphs (the links between the same nodes are not recorded repeatedly), and the results are shown in Table 3 and Fig. 2.

It can be obtained from Table 3 and Fig. 2: After the application of the traction link algorithm, the saved calculation amount is up to 77 percent of the original data amount, at least 50 percent of the original data amount, and the average saved amount reaches 66 percent.

Therefore, the use of the traction link algorithm can greatly reduce the number of links that need to be processed without affecting the subsequent path selection results. It can save a large amount of calculation and improve the efficiency of the link weight calculation algorithm.

Comparative experiment.

At present, the traditional method with the best performance (delay and rescheduling rate) is the QoS-oriented SDN global traffic scheduling algorithm proposed in literature[28,29]. The machine learning algorithm with the smallest delay is the SDN global multi-path traffic scheduling algorithm based on reinforcement learning proposed in literature[30].

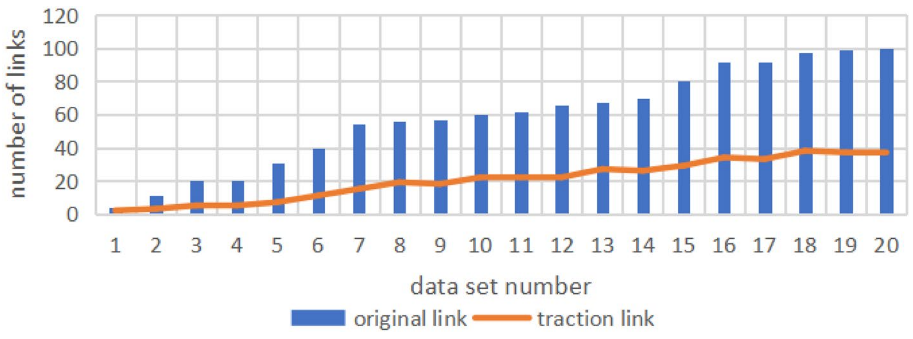

Figure 2. The extraction result of traction link.

	Delay(s)		
Data set	QOGMP	GMPRL	Pre-QOGMP
1	62	63	79
2	70	70	83
3	62	66	90
4	69	66	95
5	81	81	97
6	79	71	107
7	87	81	125
8	92	90	110
9	95	95	121
10	106	103	127
11	110	98	134
12	110	102	140
13	103	111	153
14	106	110	166
15	122	122	182
16	122	128	198
17	145	137	225
18	164	160	242
19	182	168	266
20	190	184	276

Table 4. The comparative experiment result-delay.

Compared with the algorithm proposed in literature[28,29], QOGMP algorithm considers multi-path scheduling and has a higher link utilization rate. It uses machine learning algorithms to speed up the calculation of weights, which is suitable for big data environments. In contrast, QOGMP algorithm has obvious advantages, so it is no longer verified by comparison experiments.

We evaluate the performance of QOGMP on the built simulation system. Indicators for performance evaluation include delay and rescheduling rate. We compare QOGMP with the traffic scheduling algorithm that does not use traction links (that is, implemented in the order of Algorithm 1 and Algorithm 3, hereinafter referred to as pre-QOGMP) and the algorithm proposed literature[30] (hereinafter referred to as GMPRL).

Delay here refers to the algorithm running time. We carry out comparative experiments on the three algorithms. In order to reduce the experimental error, each experiment needs to be measured multiple times to record the shortest running time.

For the delay indicator, the comparative experiment is completed on 20 different traffic views, and the experimental result is shown in Table 4.

Plot Table 4 as Fig. 3. Analysis of Fig. 3 shows that: (1) the delay of QOGMP is always lower than that of pre-QOGMP; (2) the delay of QOGMP is not much different from that of GMPRL.

It can be seen from the experimental results that QOGMP is better than pre-QOGMP and is almost consistent with GMPRL in terms of delay indicator. (1) Since both QOGMP and GMPRL use machine learning algorithms, the delay of QOGMP is not much different from that of GMPRL; (2) QOGMP increases the step of pulling link extraction which leads to the increase of running time. But at the same time, the introduction of traction link greatly saves the amount of computation. So QOGMP outperforms pre-QOGMP in delay.

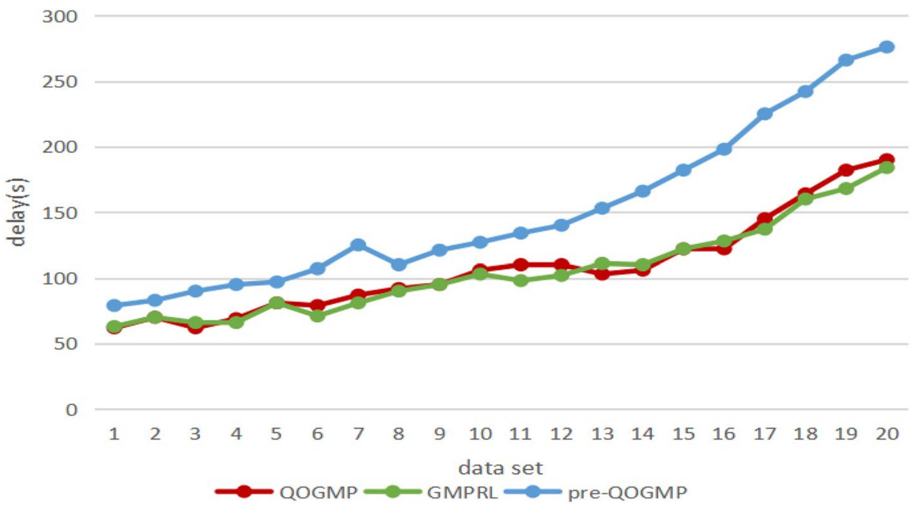

Figure 3. The comparative experiment result-delay.

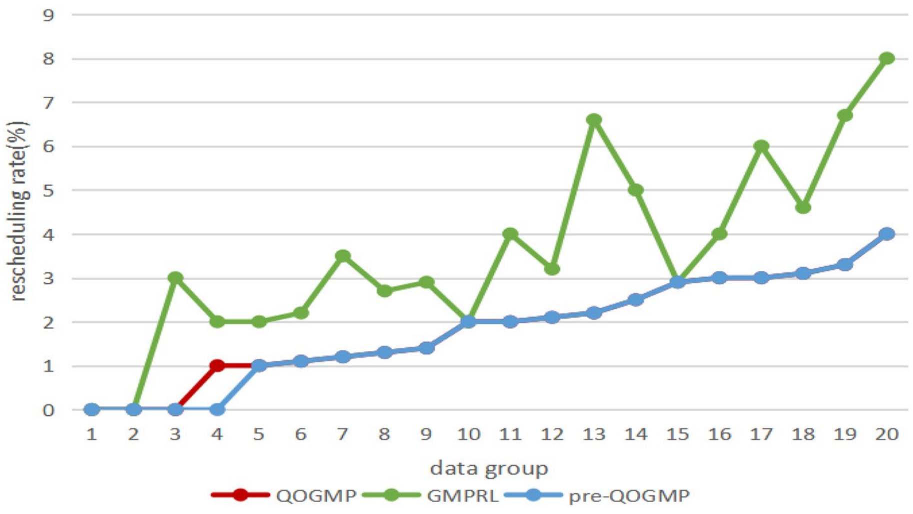

Figure 4. The comparative experiment result-rescheduling rate.

Since the three algorithms are all multi-path algorithms, the criteria used for rescheduling judgment are the same, that is, the sum of all path traffic in the solution generated by the algorithm is less than the service traffic or the transmission delay of a single path exceeds the delay tolerance.

For the indicator of rescheduling rate, we conduct 20 groups of comparative experiments, each of which was completed on 20–100 different data sets. The experimental result is shown in Table 5.

Plot Table 5 as Fig. 4. Analysis of Fig. 4 shows that: (1) The rescheduling rate of QOGMP is always not higher than GMPRL. In 17/20 cases, the rescheduling rate of QOGMP is lower than that of GMPRL, and in 3/20 cases, the rescheduling rate of QOGMP is the same as that of GMPRL. (2) The rescheduling rate of QOGMP is almost the same as that of pre-QOGMP, and the rescheduling rate of QOGMP is slightly higher than that of pre-QOGMP only in 1/20 cases.

It can be seen from the experimental results that QOGMP is better than GMPRL and is almost consistent with pre-QOGMP in terms of rescheduling rate indicator. (1) GMPRL does not consider QoS requirements, resulting in an increased probability that the traffic scheduling scheme is not suitable for service traffic, so QOGMP outperforms GMPRL in re-scheduling rate; (2) Since the introduction of traction links will not have a great impact on the final scheme, so QOGMP is almost identical to pre-QOGMP in rescheduling rate.

To sum up, compared with pre-QOGMP, QOGMP has lower delay and almost the same rescheduling rate; compared with GMPRL, QOGMP has lower rescheduling rate and almost the same delay. Therefore, our proposed QOGMP is better than GMPRL and pre-QOGMP for delay and rescheduling rate indicators.

Data group	Rescheduling rate (percent)		
	QOGMP	GMPRL	pre-QOGMP
1	0	0	0
2	0	0	0
3	0	3	0
4	1	2	0
5	1	2	1
6	1.1	2.2	1.1
7	1.2	3.5	1.2
8	1.3	2.7	1.3
9	1.4	2.9	1.4
10	2	2	2
11	2	4	2
12	2.1	3.2	2.1
13	2.2	6.6	2.2
14	2.5	5	2.5
15	2.9	2.9	2.9
16	3	4	3
17	3	6	3
18	3.1	4.6	3.1
19	3.3	6.7	3.3
20	4	8	4

Table 5. The comparative experiment result-rescheduling rate.

Conclusion

We design and implement a QoS-oriented SDN global multi-path traffic scheduling algorithm, referred to as QOGMP. Aiming at the four problems currently existing in the research of SDN control layer traffic scheduling, QOGMP has adopted solutions: (1) QoS problem is solved by using the three QoS-related network parameters of packet loss, delay, and link capacity to generate traffic scheduling scheme; (2) The easy congestion problem is alleviated by utilizing the controller global view; (3) The high latency problem is alleviated by introducing traction links and deep reinforcement learning; (4) The single path problem is solved by multi-path scheduling. We carry out comparative experiments in the built simulation environment. The experimental results show that QOGMP has better performance than the two comparison algorithms in terms of delay and rescheduling rate.

However, we still have room for improvement in terms of application scenario expansion, application algorithm improvement, and system details reproduction. The details are as follows.

(1) We only considered the three common QoS parameters of delay, packet loss and link capacity. However, the complex network environment also contains other QoS parameters such as jitter. We can conduct in-depth research on QoS parameters and improve the algorithm proposed in this article in order to further cope with the complex network environment and business requirements.
(2) As the neural network is not the innovative point and focus of the QOGMP algorithm, the neural network used in this article is the most basic. It can be replaced with an improved neural network. For example, in order to improve the update stability of neural network, we can use the target network method proposed in literature[34].
(3) The delay in the comparative experiment is up to 276 s, which is caused by hardware limitations. Later, the delay can be shortened to meet practical application requirements through Brax accelerator hardware[35].

Data availability

All data generated or analysed during this study are included in this published article.

Received: 18 June 2022; Accepted: 22 August 2022
Published online: 26 August 2022

References

1. Shu, Z. *et al.* Traffic engineering in software-defined networking: Measurement and management. *IEEE Access* **4**, 3246–3256. https://doi.org/10.1109/ACCESS.2016.2582748 (2016).
2. Zainab, Z. *et al.* Will SDN be part of 5g?. *IEEE Commun. Surv. Tutor.* **20**, 3220–3258. https://doi.org/10.1109/COMST.2018.2836315 (2018).
3. Awduche, G., Chiu, A., Elwalid, A., Widjaja, I. & Xiao, X. Overview and principles of internet traffic engineering. *IETF RFC.*https://doi.org/10.1007/BF03055356 *(2002).*

4. Manish, P., Deepti, S. & Omprakash, T. Controllers in SDN: A review report. *IEEE Access*.https://doi.org/10.1109/ACCESS.2018.2846236 *(2018)*.
5. Open networking summit. http://opennetsummit.org/ (2012).
6. McKeown, N. Software-defined networking. In *Proceeding of the INFOCOM Key Note*. https://doi.org/10.1109/CC.2014.6821732 (2009).
7. Sandra, S. H., Sriram, N. & Sakir, S. A survey of security in software defined networks. *IEEE Commun. Surv. Tutor.* **18**, 623–654. https://doi.org/10.1109/COMST.2015.2453114 (2016).
8. Ze, Y. & Kwan, L. Y. SDN candidate selection in hybrid IP/SDN networks for single link failure protection. *ACM Trans. Netw.*https://doi.org/10.1109/TNET.2019.2959588 *(2020)*.
9. Zhou, T., Cai, Z. & Xia, J. Traffic engineering for software defined networks. *Ruan Jian Xue Bao. J. Softw.* **27**, 394–417 (2016).
10. McKeown, N. *et al.* Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **38**, 69–74 (2008).
11. Chen, J. J. *et al.* Realizing dynamic network slice resource management based on SDN networks. *Int. Conf. Intell. Comput. Emerg. Appl. ICEA*.https://doi.org/10.1109/ICEA.2019.8858288 *(2019)*.
12. Gringeri, S., Bitar, N. & Xia, T. Extending software defined network principles to include optical transport. *IEEE Commun. Mag.* **51**, 32–40. https://doi.org/10.1109/MCOM.2013.6476863 (2013).
13. Hopps, C. E. Analysis of an equal-cost multi-path algorithm. http://tools.ietf.org/html/rfc2992 (2000).
14. Chiang, Y. R., Ke, C. H., Yu, Y. S., Chen, Y. & Pan, C. A multipath transmission scheme for the improvement of throughput over SDN. *Int. Conf. Appl. Syst. Innov.*https://doi.org/10.1109/ICASI.2017.7988122 *(2017)*.
15. Jiang, Z., Wu, Q., Li, H. & Wu, J. scmptcp: Sdn cooperated multipath transfer for satellite network with load awareness. *IEEE Access*.https://doi.org/10.1109/ACCESS.2018.2820719 *(2018)*.
16. Barakabitze, A. A., Sun, L. F., Mkwawa, I. H. & Ifeachor, E. A novel QOE-centric SDN-based multipath routing approach for multimedia services over 5g networks. *IEEE Int. Conf. Commun.*https://doi.org/10.1109/ICC.2018.8422617 *(2018)*.
17. Manan, D. & Aayush, K. Multi-constraint QOS disjoint multipath routing in SDN. *Moscow Workshop Electron. Netw. Technol.*https://doi.org/10.1109/MWENT.2018.8337305 *(2018)*.
18. Zhu, Q. B. *Research and Implementation of Load Balancing Traffic Scheduling Based on SDN*. (Wuhan Research Institute of Posts and Telecommunications, 2018).
19. Lei, M. *Research on Traffic Scheduling Algorithm Based on SDN Data Center*. (Xi'an Technological University, 2018).
20. Sheu, J. P., Liu, L. W., Jagadeesha, R. B. & Chang, Y. An efficient multipath routing algorithm for multipath TCP in software-defined networks. *Eur. Conf. Netw. Commun.*https://doi.org/10.1109/EuCNC.2016.7561065 *(2016)*.
21. Syed, A. H., Shuja, A. & Imran, R. A dynamic multipath scheduling protocol (DMSP) for full performance isolation of links in software defined networking (SDN). In *2nd Workshop on Recent Trends in Telecommunications Research*. https://doi.org/10.1109/RTTR.2017.7887866 (2017).
22. Muhammad, M. U., Izaz, A. K. & Syed, A. A. S. Delay-efficient forwarding in SDN assisted mesh networks: An application of shapley value. In *13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics*. https://doi.org/10.1109/MACS48846.2019.9024804 (2019).
23. Wang, Q. T. *et al.* Implementation of multipath network virtualization scheme with SDN and NFV. In *IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications*. https://doi.org/10.1109/PIMRC.2017.8292349 (2017).
24. Sheu, R. T. & Wu, J. L. C. Performance analysis of rate control with scaling QOS parameters for multimedia transmissions. In *IEEE Proceedings Communications*. https://doi.org/10.1049/ip-com:20030714 (2003).
25. Srinivasan, D. S-flink: Schedule for QOS in flink using SDN. In *IEEE 40th Annual Computer Software and Applications Conference*. https://doi.org/10.1109/COMPSAC.2016.190 (2016).
26. Mao, Y. *Research on SDN-Based Traffic Scheduling Technology*. (Southwest Jiaotong University, 2018).
27. Egilmez, H. E., Dane, S. T., Bagci, K. T. & Tekalp, A. Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In *Proceeding of the Signal and Information Processing Association Annual Summit and Conference* 1–8 (2012).
28. Ongaro, F., Cerqueira, E., Foschini, L., Corradi, A. & Gerla, M. Enhancing the quality level support for real-time multimedia applications in software-defined networks. In *Proceeding of the International Conference on Computing, Networking and Communications* 505–509. https://doi.org/10.1109/ICCNC.2015.7069395 (2015).
29. Ongaro, F. *Enhancing Quality of Service in Software-Defined Networks* (UNIBO, 2014).
30. Truong, T. H., Ngo, D. D. K., Nguyen, X. D. & Thanh, N. A global multipath load-balanced routing algorithm based on reinforcement learning in SDN. In *International Conference on Information and Communication Technology Convergence*. https://doi.org/10.1109/ICTC46691.2019.8939987 (2019).
31. Reza, M., Sobouti, M. J., Raouf, S. & Javidan, R. Network traffic classification using machine learning techniques over software defined networks. *Int. J. Adv. Comput. Sci. Appl.*. https://doi.org/10.14569/IJACSA.2017.080729 (2017).
32. Gertsiy, A. & Rudyk, S. Analysis of quality of service parameters in IP-networks. In *The Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology*. https://doi.org/10.1109/INFOCOMMST.2016.7905340 (2016).
33. Sun, P., Lan, J., Guo, Z., Xu, Y. & Hu, Y. Improving the scalability of deep reinforcement learning-based routing with control on partial nodes. In *IEEE International Conference on Acoustics, Speech and Signal Processing* 3557–3561. https://doi.org/10.1109/ICASSP40776.2020.9054483 (2020).
34. Cisco. Intent-Based Networking. https://www.cisco.com/.
35. Freeman, C., Frey, E. & Raichuk, A. *Brax—A Differentiable Physics Engine for Large Scale Rigid Body Simulation*. (Computer Science, 2021).

Author contributions
Y.G. conducted experiments and wrote the first draft, G.H. read the literature and completed the initial study design, D.S. conducted experiments and analyzed the data. All authors reviewed the manuscript.

Competing interests
We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted. All authors have approved the manuscript and agree with its submission. The corresponding author is responsible for submitting a competing interests statement on behalf of all authors of the paper.

Additional information
Correspondence and requests for materials should be addressed to Y.G.