Postprint: Scheduling Mechanism for Load Balancing in Edge Computing Networks

Authors: Dong Qian, Ma Yuxiang, Li Jun

Date: 2019-01-03T00:00:00+00:00

Abstract

In edge computing application scenarios, resource deployment and allocation constitute critical issues. To address load balancing requirements in edge computing networks, we propose a centralized control-based scheduling mechanism. First, we determine at which network nodes to deploy edge computing functions; then, for user data and requests, we strive to minimize the average end-to-end latency of traffic through scheduling while satisfying relevant load balancing constraints. Evaluation results demonstrate that the number of edge computing nodes, as well as the load balancing degree of both computing and network resources, may significantly impact the average end-to-end latency of traffic. By selecting only a small number of appropriate nodes as edge computing nodes and adjusting the load balancing of computing and network resources to an appropriate level, the average end-to-end latency can be effectively reduced.

Full Text

Preamble

Load Balancing Oriented Scheduling Scheme in Edge Computing Network

Dong Qian[1],[2],[3], Ma Yuxiang[1],[2], Li Jun[1]†

(1. Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China;
2. University of Chinese Academy of Sciences, Beijing 100049, China;
3. Foshan University, Foshan, Guangdong 528000, China)

Abstract: In edge computing application scenarios, resource deployment and allocation are critical issues. To address load balancing requirements in edge computing networks, this paper proposes a centralized control-based scheduling mechanism. The scheme first determines which network nodes should deploy

edge computing functions, then schedules user data and requests to minimize the average end-to-end delay of traffic while satisfying relevant load balancing constraints. Evaluation results demonstrate that the number of edge computing nodes and the load balancing degree of computing and network resources can both affect the average end-to-end delay. Selecting a small number of suitable nodes as edge computing nodes and appropriately balancing the load of computing and network resources can effectively reduce the average end-to-end delay.

Keywords: edge computing; centralized control; load balancing; segment routing

0 Introduction

As user demands become increasingly diverse and network traffic continues to grow, edge computing emerges as a key technology for cloud computing and 5G networks. Edge computing addresses the requirements of cloud computing and IoT/mobile network scenarios by deploying computing functions at network nodes between end devices and cloud data centers. This approach reduces data center burden, saves energy, and improves data processing real-time performance [1]. Figure 1 [Figure 1: see original paper] illustrates the edge computing network architecture.

The network is primarily divided into access and core layers [2]. The access layer connects user devices and deploys edge computing nodes, represented as clouds in Figure 1. Note that Figure 1 omits interconnections between access layer nodes, between access layer nodes and user devices, and between user devices (if any). The core layer connects the access layer and data centers, represented as ellipses in Figure 1, with solid lines indicating interconnection links between core layer nodes, access layer nodes, and data centers. Each edge computing node possesses computing and storage capabilities. Typically, after preprocessing user data and requests, the traffic that needs forwarding to data centers is significantly reduced [1].

Consequently, determining which network nodes should deploy edge computing functions and how to schedule user data and requests while balancing node loads becomes a crucial consideration for edge computing networks. Software-defined networking (SDN) employs centralized control mechanisms with a global network view and the ability to control node operations, offering outstanding advantages for edge computing networks [3]. This paper proposes a centralized control-based scheduling mechanism for load balancing in edge computing networks. The main contributions are:

a) An analysis of the scheduling model in edge computing networks and the proposal of a centralized control-based, load balancing-oriented scheduling mechanism;

b) A method to determine which specific network nodes should deploy edge computing functions;

c) A method to determine how to schedule user data and requests to minimize average end-to-end delay while satisfying load balancing constraints.

1 Related Work

In recent years, edge computing has garnered widespread attention, with increasing research efforts addressing its applications in mobile networks, IoT, vehicular networks, and resource scheduling problems [4-6]. Emerging network technologies such as software-defined networking and segment routing are gradually being applied to edge computing [3,7,8].

Shi et al. [1] summarized edge computing concepts and principles, introduced representative application examples, and outlined major challenges. Mao et al. [9] focused on mobile edge computing, discussing research progress and challenges in system deployment, caching mechanisms, mobility management, energy efficiency, and privacy, while also introducing standardization efforts and typical application scenarios. Mach et al. [10] examined computation offloading use cases in mobile edge computing, summarizing research status for full offloading and partial offloading scenarios. Full offloading primarily aims to minimize execution delay, minimize energy consumption under delay constraints, or balance delay and energy consumption. Partial offloading focuses on minimizing energy consumption under delay constraints or achieving a balance between delay and energy consumption, considering appropriate computing resource allocation and distinguishing between single-node and multi-node computation allocation. Sun et al. [11] proposed PRIMAL, which considers end-to-end delay between users and computing nodes and migration costs of computing and storage resources to balance end-to-end delay and migration costs. Wang et al. [12] studied load placement problems in edge computing to determine how to place applications on edge computing nodes. These works demonstrate that resource deployment and allocation are important issues in edge computing scenarios, with end-to-end delay between user devices and edge computing nodes being a primary optimization objective. However, few studies have examined how load balancing requirements for computing and network resources affect end-to-end delay.

On the other hand, Baktir et al. [3] proposed integrating software-defined networking with edge computing, where controllers manage data flows and perform service orchestration and other management tasks, typically including service discovery, service debugging and migration, performance tuning and optimization, and user handover modules. The performance tuning and optimization module primarily focuses on network and computing resource usage, managing loads on edge computing nodes. Filsfils et al. [13] proposed segment routing, which enables flexible configuration of data flow forwarding paths while reducing

the burden of forwarding rule distribution on controllers. Hartert et al. [14] proposed the centralized optimizer DEFO for controlling forwarding paths, where segment routing's flexibility and scalability play important roles. Desmouceaux et al. [15] proposed 6LB, which uses IPv6-based segment routing technology to guide packets and designs a load balancer. These works show that combining centralized control mechanisms with edge computing can optimize load balancing and network performance, while segment routing supports flexible path control and enhances controller scalability.

Therefore, this paper treats load balancing requirements as constraints and aims to minimize average end-to-end delay through scheduling based on centralized control and segment routing technologies.

2 Overall Architecture and Problem Analysis

Based on abstraction of real-world scenarios and the design philosophy of this mechanism, we assume all network nodes below the core layer in Figure 1 support segment routing technology. Segment routing forwarding rules only need configuration at source nodes, enhancing centralized controller scalability. We categorize these network nodes into three types:

a) User devices: In addition to basic network node functions, they send data and requests to edge computing nodes;

b) Edge computing nodes: In addition to basic network node functions, they process user device data and requests, possess computing and storage resources, and return processed data and responses to user devices or send them to data centers;

c) Intermediate nodes: Nodes that are neither user devices nor have edge computing capabilities, possessing only basic network node functions.

This mechanism includes a centralized controller in addition to network nodes. Using a common network topology as an example with 6 user devices, 3 edge computing nodes, and 3 intermediate nodes, the overall architecture and controller functions are illustrated in Figure 2 [Figure 2: see original paper]. The controller only needs access to the edge computing network to use southbound interface protocols such as NETCONF [16] to obtain network and resource usage information and configure devices (this function is not shown in Figure 2). Solid lines with unidirectional arrows indicate the controller selecting appropriate network nodes to deploy edge computing functions. Dashed lines with arrows indicate the controller receiving scheduling demands from user devices and distributing segment routing forwarding rules to user devices and edge computing nodes. Solid lines between network nodes represent their interconnection links.

When edge computing nodes are determined and user devices generate data and requests, the processing steps are as follows:

a) User devices send traffic information of data and requests to the controller;
b) Based on current network and resource usage, the controller calculates which edge computing nodes should receive the traffic and which paths should be used, then distributes segment routing forwarding rules to user devices and notifies relevant edge computing nodes to reserve resources. If processed data and responses need to be sent back to user devices, the controller similarly calculates appropriate forwarding paths based on current conditions and distributes segment routing forwarding rules to edge computing nodes;
c) After receiving the controller's response, user devices send traffic according to the distributed rules;
d) Edge computing nodes process received traffic and, after receiving the controller's response, send traffic according to the distributed rules. User devices eventually receive processed data and responses from edge computing nodes. Figure 3 [Figure 3: see original paper] illustrates these processing steps.

In Figure 3, the dashed line with a unidirectional arrow between user devices and the controller corresponds to the dashed line with bidirectional arrows in Figure 2, representing user devices sending scheduling demands and the controller distributing forwarding rules. The dashed line with a unidirectional arrow between the controller and edge computing nodes corresponds to that in Figure 2, representing the controller distributing forwarding rules. Thick unidirectional arrows represent traffic exchange between user devices and edge computing nodes. Intermediate nodes are omitted in Figure 3 as they support segment routing technology, requiring no forwarding rule distribution from the controller.

Network topology is typically represented as a directed graph. Network nodes are denoted by i, with the set of all core layer nodes denoted by N. Links are directed, denoted by e, with the set of links denoted by E. Each node i belongs to exactly one of the three categories defined above. For clarity, user devices and edge computing nodes are also denoted by s and t respectively, with sets S and T. The processing capacity of computing resources at node t is denoted by v_t, with units consistent with user device sending rates (bps) for convenience.

Assume network topology information is known, S is known, and T is determined through the method in Section 3.1. Let $|T|$ denote the number of elements in T.

For simplicity, assume a user device s has traffic from only one application (easily extendable to multiple applications), with sending rate $d_s > 0$. The portion sent to a particular t is $d_s(t)$. The maximum computing resource utilization is set to λ ($0 \leq \lambda \leq 1$). The following constraints apply:

Equation (1) specifies the value requirements for $d_s(t)$ when computation is allowed to be distributed across multiple nodes versus restricted to a single node. Equation (2) ensures all d_s is sent. Equation (3) ensures traffic received by each t can be processed by its actually usable computing resources. Clearly, with constraints (1) and (2) satisfied, a smaller λ indicates a narrower possible

range of computing resource utilization across all t, so λ is used to measure the permitted load balancing degree of computing resources. Equation (4) ensures that even with only one edge computing node, all traffic from s can be processed when $\lambda = 1$, meaning $|T|$ can be as small as 1, guaranteeing network robustness.

After processing $d_s(t)$, edge computing node t returns $r_t(s)$ to s. For simplicity, this paper assumes all traffic from s is of the same type (easily extendable to multiple types). The traffic type determines coefficient α, where generally a smaller α (even 0) indicates the traffic is primarily data requiring processing, while a larger α indicates the traffic is primarily requests requiring processing.

Given network topology information, candidate loop-free forwarding paths between two nodes can be precomputed based on preset conditions and segment routing rules. A candidate forwarding path is denoted by p, with p_e indicating whether link e is on path p (1 if yes, 0 otherwise). The candidate path sets from s to $i \in N \quad S$ and from $i \in N \quad S$ to s are denoted by P_{si} and P_{is} respectively. Path hop count h_p measures end-to-end delay when path p is used. For each pair (s, i), paths in P_{si} and P_{is} correspond in opposite directions, so the minimum hop counts are identical, denoted by h_{si}. Let w_p denote path p's routing cost (weight), and w_{si} denote the weight of the minimum-hop path. If traffic x_p is carried on path p, then:

Equations (6) and (7) indicate that $d_s(t)$ and $r_t(s)$ have been assigned forwarding paths. Notably, when computation is restricted to a single node, this paper typically requires using only one forwarding path to avoid packet reordering [17]. From equations (1)-(2) and (5)-(7), we have:

When computation distribution across multiple nodes is allowed, there is no such restriction:

Finally, the average hop count H of traffic is defined as:

The H calculated by equation (11) measures the average end-to-end delay of traffic. If network resources are also considered, with link capacity denoted by c_e and maximum link utilization set to θ ($0 \leq \theta \leq 1$), then similar to λ, θ measures the permitted load balancing degree of network resources:

Equation (12) indicates that actually usable network resources may constrain forwarding path selection, thereby affecting the obtained H. Typically, under identical conditions for α, λ, and θ, the obtained H should be minimized, with the optimization task written as minimize H. Specifically, if network resource load balancing constraints are ignored and the optimization task is minimize H, the h_p of selected forwarding paths from s to t and t to s should equal h_{st}. In this case, only equations (1)-(3) need consideration, and equation (11) simplifies to:

Equation (13) shows that α does not affect the obtained H in this scenario.

3.1 Edge Computing Node Selection

Before selecting edge computing nodes, known information includes network topology, set S with corresponding d_s, and for all s to all $i \in N \ S$, the values of h_{si} and w_{si}. During selection, computing and network resource load balancing constraints are ignored, with the optimization task being minimize H. Since T is unknown, we first set the final $|T|$ equal to k.

Let u_i indicate whether node $i \in N \ S$ is selected as t (1 if selected, 0 otherwise). Let $d_s(i)$ denote the portion of traffic from s sent to i. Set a sufficiently large number m, ignoring units and taking only the numerical value, typically ensuring $m > \max\{\max_{s\in S, i\in N\ S}\{h_{si}\}, \max_{s\in S, i\in N\ S}\{w_{si}\}\}$. For node $i \in T$, since it is typically denoted by t, the corresponding P_{si}, P_{is}, h_{si}, and w_{si} can also be denoted by P_{st}, P_{ts}, h_{st}, and w_{st}. Let path $p \in P_{st} \cup P_{ts}$.

Equation (14) is the optimization task for the solver during edge computing node selection. Its practical significance is: under the premise of minimizing H, use m to ensure that when multiple solutions yield the same minimum H, the solution with minimal routing cost is selected as the final solution. Equation (15) derives from the definition of $d_s(i)$ and also indicates that $d_s(i) = 0$ when i is not selected as t. Equation (16) rewrites equation (2). Equation (17) ensures the final $|T| = k$. Solving can use optimization solvers like Gurobi [18], with the optimization task being equation (14) considering only equations (15)-(17). After obtaining T, for $i \in T$ (denoted by t), $d_s(i)$ and h_{si} become $d_s(t)$ and h_{st}, and H is calculated by equation (13).

Alternatively, a pending set S' can be set for S, with two temporary variables O' and O'', using the following algorithm:

Algorithm 1: Edge Computing Node Selection

1) Input network topology, S with corresponding d_s, k, m, and all h_{si}, w_{si} for s to $i \in N \ S$
2) Initialize T as empty set, S' as S
3) for temp = 1 to k do
4)
5) Initialize candidate edge computing nodes as empty
6) for each $i \in N \ (S \cup T)$ do
7)
8)
9)
10)
11)

12)
13)
14) Add candidate edge computing nodes to T

15) for each $s \in S'$ do

16)
17)
18)
19)
20)
21) if S' is empty then

22)
23)
24)
25) end for

26)
27) if $|T| < k$ then

28) Randomly select $(k - |T|)$ nodes from $N \quad (S \cup T)$ and add to T

29) else

30) end if

The algorithm outputs T and H with $|T| = k$. Edge computing nodes are configured with computing resources according to T and corresponding v_t.

3.2 Scheduling Computation

After node selection, the controller performs scheduling computation. Before each computation, values for α, λ, and θ should be set. Generally, more conditions increase computational difficulty.

If network resource load balancing constraints are ignored, the optimization task is minimize H, considering only equations (1)-(3), with H calculated by equation (13). Solving can use optimization solvers or the following algorithm:

Algorithm 2: Scheduling Computation

a) Input S with corresponding d_s, T with corresponding v_t, λ, m, and all h_{st}, w_{st} for s to t

b) Initialize all $d_s(t)$ to 0. Process all s in descending order of d_s. When calculating current s's $d_s(t)$, only consider already processed s and current s, minimizing current s's $\sum_{t \in T} d_s(t) \cdot h_{st}$ while satisfying equations

(1)-(3). Keep processed s's $d_s(t)$ unchanged in subsequent calculations. After processing all s, if a solution exists, temporarily store it as Solution 1.

c) Initialize all $d_s(t)$ to 0. Process all s in ascending order of d_s. When calculating current s's $d_s(t)$, only consider already processed s and current s, minimizing current s's $\sum_{t\in T} d_s(t) \cdot h_{st}$ while satisfying equations (1)-(3). Keep processed s's $d_s(t)$ unchanged in subsequent calculations. After processing all s, if a solution exists, temporarily store it as Solution 2.

d) If both steps b) and c) have solutions, compare Solution 1 and Solution 2, calculate their H values using equation (13), and select the one with smaller H as the final solution. If only one has a solution, select that solution and its H.

The algorithm outputs all $d_s(t)$ and H. For each non-zero $d_s(t)$, the controller selects a forwarding path $p \in P_{st}$ with $h_p = h_{st}$, setting $x_p = d_s(t)$. Calculate $r_t(s)$ from $d_s(t)$, and for each non-zero $r_t(s)$, select a forwarding path $p \in P_{ts}$ with $h_p = h_{st}$, setting $x_p = r_t(s)$. Relevant configurations are distributed to user devices and edge computing nodes based on x_p.

If network resource load balancing constraints are also considered, the optimization task is minimize H, considering equations (1)-(3), (5)-(10), and (12), with H calculated by equation (11). Solving uses optimization solvers, outputting all x_p. Only configurations corresponding to $x_p > 0$ are distributed to user devices and edge computing nodes.

4 Experimental Evaluation and Analysis

This paper obtains experimental network topology information from the public dataset SNDLib [19], which contains 22 nodes. The preset S includes 10 of these nodes. All link capacities are 40,000 Mbps. Three traffic matrices (TMs) are selected from SNDLib's TM dataset. For each TM, the sum of outgoing traffic from these 10 s nodes is calculated as their respective d_s values. The resulting S and corresponding d_s from these three TMs form Scenarios 1, 2, and 3. All v_t are set to 40,000 Mbps.

First, all P_{si} and P_{is} between each s and each $i \in N \ \ S$ are computed. For each pair (s, i), P_{si} and P_{is} contain paths satisfying SLD [20] (segment list depth) ≤ 2 and loop-free conditions [21], from which all h_{si} and w_{si} are obtained for subsequent calculations. Edge computing node selection is performed without considering computing and network resource load balancing constraints. For Scenario 1, varying k values are used to compare results from the optimization solver and Algorithm 1, as shown in Figure 4 [Figure 4: see original paper].

Figure 4 shows that for the same k, the H values obtained by the optimization solver and Algorithm 1 are identical. Since the optimization solver's H is the minimum under corresponding conditions, Algorithm 1's H is also minimal. When $k \leq 4$, larger k yields smaller H; when $k \geq 4$, the obtained H

remains the same. This occurs because when $k < 4$, some s have $\min_{t \in T}\{h_{st}\} > \min_{i \in N\ S}\{h_{si}\}$. When k increases to 4 or more, all s have $\min_{t \in T}\{h_{st}\} = \min_{i \in N\ S}\{h_{si}\}$, so adding more t cannot yield a smaller H. Additionally, when $k = 1, 2, 3, 4$, the T obtained by both methods are identical. Algorithm 1 is a relatively simpler greedy algorithm. These results demonstrate that using Algorithm 1 to select a small number of suitable nodes as edge computing nodes can effectively reduce average hop count while simplifying computation.

Next, k is set to 4. Based on Scenario 1, Algorithm 1 computes T, yielding all h_{st} and w_{st} for s to t. SN denotes restricting computation to single nodes, while MN denotes allowing multiple nodes. Scheduling computation is performed without network resource load balancing constraints. For Scenarios 1, 2, and 3, varying λ values are used to compare results from the optimization solver and Algorithm 2, as shown in Figure 5 [Figure 5: see original paper].

Figure 5 shows that for small λ, the H from the optimization solver is slightly smaller than Algorithm 2's H when computation can be distributed across multiple nodes, and significantly smaller or equal when restricted to single nodes. For large λ, both methods yield identical H. Generally, before λ reaches a certain threshold, distributing computation across multiple nodes yields H values less than or equal to single-node restriction. Beyond this threshold, further increasing λ does not change the obtained H. This indicates that the permitted load balancing degree of computing resources can significantly affect average hop count, particularly for single-node restriction where smaller λ may substantially increase average hop count. Multi-node distribution shows relatively smaller increases. When λ is large, Algorithm 2 also achieves good results.

Finally, k remains 4. Based on Scenario 1, Algorithm 1 computes T, yielding all P_{st} and P_{ts} between each s and t. Only multi-node distribution is considered with $\lambda = 1$. Scheduling computation incorporates network resource load balancing constraints with α set to 0, 0.5, and 1. For Scenarios 1, 2, and 3, varying θ values are used with the optimization solver, as shown in Figure 6 [Figure 6: see original paper].

Figure 6 shows that before θ reaches a certain threshold, smaller θ yields larger H. Beyond this threshold, further increasing θ does not change the obtained H. For the same θ, $\alpha = 0$ or 1 yields the same H as $\alpha = 0.5$, because when s sends and receives traffic in opposite directions, if received traffic size is 0 or equals sent traffic size, the obtained H actually depends only on sent traffic. Before θ reaches the threshold and for the same θ, $\alpha = 0$ or 1 yields larger H than $\alpha = 0.5$, because when sent and received traffic are unequal and received traffic is non-zero, the larger of the two has a greater proportion of traffic using higher-hop paths due to network resource load balancing constraints. These results demonstrate that the permitted load balancing degree of network resources can also affect average hop count, with smaller θ potentially increasing average hop count.

5 Conclusion

This paper proposes a centralized control-based scheduling mechanism for load balancing requirements in edge computing networks. By scheduling to minimize average end-to-end delay while satisfying load balancing constraints, the paper analyzes the model and proposes algorithms to determine which network nodes should deploy edge computing functions and how to schedule user data and requests. Evaluation results show that the number of edge computing nodes and the load balancing degree of computing and network resources can affect average end-to-end delay. Selecting a small number of suitable nodes as edge computing nodes and appropriately balancing computing and network resource loads can effectively reduce average end-to-end delay.

Future work will consider reasonable scheduling under failures of links, shared risk link groups, and nodes, while addressing centralized controller scalability and energy efficiency needs of edge computing networks.

References

[1] Shi Weisong, Cao Jie, Zhang Quan, et al. Edge computing: vision and challenges [J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646.
[2] Byers C C. Architectural imperatives for fog computing: use cases, requirements, and architectural techniques for FOG-enabled IoT networks [J]. IEEE Communications Magazine, 2017, 55(8): 14-20.
[3] Baktir A C, Ozgovde A, Ersoy C. How can edge computing benefit from software-defined networking: a survey, use cases, and future directions [J]. IEEE Communications Surveys & Tutorials, 2017, 19(4): 2769-2795.
[4] Zhao Zhiwei, Min Geyong, Gao Weifeng, et al. Deploying edge computing nodes for large-scale IoT: a diversity aware approach [J]. IEEE Internet of Things Journal, 2018, 5(5): 3606-3614.
[5] Zuo Yuan, Wu Yulei, Min Geyong, et al. Learning-based network path planning for traffic engineering [J]. Future Generation Computer Systems, 2019, 92: 59-67.
[6] Ma Yuxiang, Wu Yulei, Ge Jingguo, et al. An architecture for accountable anonymous access in the Internet-of-Things network [J]. IEEE Access, 2018, 6: 14451-14461.
[7] Cheng Xiangle, Wu Yulei, Min Geyong, et al. Network function virtualization in dynamic networks: a stochastic perspective [J]. IEEE Journal on Selected Areas in Communications, 2018.
[8] Li Jianhua, Jin Jiong, Yuan Dong, et al. EHOPES: data-centered fog platform for smart living [C]//Proc of International Telecommunication Networks and Applications Conference. 2015: 147-154.
[9] Mao Yuyi, You Changsheng, Zhang Jun, et al. A survey on mobile edge computing: the communication perspective [J]. IEEE Communications Surveys & Tutorials, 2017, 19(4): 2322-2358.
[10] Mach P, Becvar Z. Mobile edge computing: a survey on architecture and computation offloading [J]. IEEE Communications Surveys & Tutorials, 2017,

19(3): 1628-1656.
[11] Sun Xiang, Ansari N. PRIMAL: profit maximization avatar placement for mobile edge computing [C]//Proc of IEEE International Conference on Communications. 2016: 1-6.
[12] Wang Shiqiang, Zafer M, Leung K K. Online placement of multi-component applications in edge computing environments [J]. IEEE Access, 2017, 5: 2514-2533.
[13] Filsfils C, Nainar N K, Pignataro C, et al. The segment routing architecture [C]//Proc of IEEE Global Communications Conference. 2015: 1-6.
[14] Hartert R, Vissicchio S, Schaus P, et al. A declarative and expressive approach to control forwarding paths in carrier-grade networks [J]. ACM SIGCOMM Computer Communication Review, 2015, 45(4): 15-28.
[15] Desmouceaux Y, Pfister P, Tollet J, et al. 6LB: scalable and application-aware load balancing with segment routing [J]. IEEE/ACM Transactions on Networking, 2018, 26(2): 819-834.
[16] Enns R, Bjorklund M, Schoenwaelder J, et al. RFC 6241, Network configuration protocol (NETCONF) [S]. IETF 2011.
[17] Kandula S, Katabi D, Sinha S, et al. Dynamic load balancing without packet reordering [J]. ACM SIGCOMM Computer Communication Review, 2007, 37(2): 51-62.
[18] Gurobi Optimization, LLC. Gurobi optimizer reference manual [EB/OL]. [2018-11-23]. http://www.gurobi.com.
[19] Orlowski S, Wessäly R, Pióro M, et al. SNDlib 1.0-survivable network design library [J]. Networks, 2010, 55(3): 276-286.
[20] Moreno E, Beghelli A, Cugini F. Traffic engineering in segment routing networks [J]. Computer Networks, 2017, 114: 23-31.
[21] Dong Qian, Li Jun, Ma Yuxiang, et al. Traffic scheduling method based on segment routing in software-defined networking [J]. Journal on Communications, 2018, 39(11): 23-35.

Note: Figure translations are in progress. See original paper for figures.