Disaster-Aware Dynamic Routing for SDN-Based Multi-Site Data Center Networks

Wenhao Zhang[1], Xiaochen Li[2], and Lisheng Ma[3]
[1]School of Systems Information Science, Future University Hakodate, 116-2
Kamedanakano-cho, Hakodate, Hokkaido, 041-8655, Japan
[2]PLA Strategic Support Force Information Engineering University, Zhengzhou, Henan, 450001, PR China
[3]School of Computer and Information Engineering, Chuzhou University, Anhui 239000, PR China

In recent years, cloud computing technology has been developing rapidly. As a result, the internal traffic of large-scale enterprises' data centers has increased significantly. It has become important to improve the disaster tolerance capability of data centers to ensure user data security. However, the data center network relies on its physical infrastructure. Large-scale disasters may damage the infrastructure and cause huge data loss or connection interruption. Software-defined network (SDN) is an innovative network architecture that separates the control and forwarding layers of the network. Thus, SDN promotes network programmability and opens up new ways to design disaster-resistant networks. Based on SDN technology, we propose a disaster-aware dynamic routing (DADR) scheme. When a disaster signal is received, the SDN controller notifies the Global Server Load Balance (GSLB) device and stops declaring the IP of the disaster-stricken data center. At the same time, the SDN controller sends the server and client session information and the current traffic information of the disaster-stricken data center to other sites, where the optimal routing path is calculated for the data center based on the traffic characteristics by the proposed Lagrangian Relaxation based Bellman-Ford Parallel algorithm (LRBFP). Our results show that in the event of a disaster, based on the proposed DADR scheme and LRBFP algorithm, the packet loss rate and network delay can be greatly reduced.

***Index Terms*—Disaster-aware, dynamic routing, data center networks, software-defined network, parallel algorithm.**

I. Introduction

A Data center is an infrastructure in a large IT enterprise. It consists of thousands of servers connected together through a data center network which can provide powerful computing and storage capabilities to support various businesses of the enterprise. With the rapid development of cloud computing, the number of online business users are sharply increased. This requires data centers to provide reliable network services for users. In general, data center provides adopt a multi-site data center network architecture to improve the availability (One type of multi-site data center network topology comprises at least a pair of data centers and both of which are active. Traffic can go to the nearest active data center to obtain the services). However, disasters (e.g. earthquake) always lead to failures of data centers [1], [2], and then services provided by disaster-stricken data centers will be affected under disaster situations. Under such situation, traffic for disaster-affected services will be rerouted to other interconnected multi-site data centers that can provide the required services. The data centers received the disaster-affected traffic may be congested due to the limited network bandwidth resources. As a result, packet loss and delay will increase which seriously affects network performance and user experience. Thus, the study of traffic scheduling in data center network under disaster scenario is critical for data center operators to guarantee the network performance [3], [4], [5].

The traffic scheduling algorithm is an effective way to solve network congestion, which can be divided into two categories, i.e., traffic scheduling among multi-site Data Centers and traffic scheduling in the Data Center. The former usually is used to a multi-site data centers layout that based on Intelligent Domain Name System (DNS) and Global Server Load Balancing (GSLB) to achieve high availability and continuity. Authors in [6] describe a system for providing distributed global server load balancing over resources across multi-site data centers, and [7] also proposes a new dynamic load balancing method that uses dynamic DNS update and round-robin mechanism without modifying the DNS list. Both of them can achieve fast recovery after a disaster, but they will cause 1-2 seconds of lag and information loss due to disasters. These previous works cannot take the early warning time of disaster into account to carry out the pre-disaster protection, which can reduce the impact for users under disasters.

The latter considers traffic scheduling in the Data Center, where Software Defined Network (SDN) [8] is adopted to achieve load balancing of the data center. The paper [9] proposes the Plug-n-Serve system implementing a load balancing algorithm called LOBUS (load-balancing over unstructured networks) by using OpenFlow for unstructured networks. The paper [10] develops a load balancing algorithm for handling multiple services (called LBMS) by SDN technology. It uses an SDN device (i.e., FlowVisor) to achieve network virtualization and coordinate multiple controllers. Each controller handles requests destined for different services. Since SDN calculates the path of each traffic in real-time in the above works, the sudden traffic scheduling under disaster scenario will bring processing pressure to the controller which leads to network performance degradation. Thus, these works cannot well adapt to the disaster scenario.

To address the above limitations, this paper assumes that multi-site data center network topology comprises a pair of

Manuscript received June 2, 2020; revised December 4, 2020. Corresponding author: Xiaochen Li (email: xchenli111@gmail.com).

data centers with SDN controller, respectively, and traffic can go to both of which to obtain service. For a disaster scenario, we develop an interface in the SDN controller which can receive the disaster signal from the disaster monitoring system in [11]. According to the disaster signal, the SDN controller notifies the Global Server Load Balance (GSLB) device to stop declaring the IP of the disaster-stricken data center, and then the traffic is blocked to enter into the disaster-stricken data center. Meanwhile, the SDN controller sends the server and client session information to another site to ensure user access without interruption. Furthermore, the SDN controller also sends the current traffic information of the disaster-stricken data center to another site at the same time when received the disaster signal. To fully utilize link resources to deal with the traffic, we propose a GPU-based Lagrangian Relaxation based Bellman-Ford Parallel algorithm (LRBFP) for the SDN controller to calculates the current optimal routing path based on traffic information characteristic. Our work can significantly reduce the packet loss rate and delay as well as shorten the disaster recovery time. Such that user perception is minimized and data security is improved when disaster scenarios.

The rest of the paper is organized as follows: The proposed scheme is introduced in Section II. In Section III we discuss the performance of our scheme. Finally, we conclude this paper in Section IV.

II. Proposal and Design

As the size of the data center network continues to expand and the number of applications increases, the number of flows transmitted in the data center network will be huge. Therefore, when the disaster happens, the network control of the traffic from the disaster data center may not be able to complete the optimization calculation of the flow routing within an acceptable time limit. Based on this, we propose a disaster-aware dynamic routing (DADR) scheme and design a Lagrangian Relaxation based Bellman-Ford Parallel algorithm (LRBFP), which includes the following modules, disaster warning reception, large and small flow monitoring, algorithm design.

A. Design of Our Scheme

This paper proposes a DADR scheme. This scheme can make full use of data center network resources to meet the needs of most users when a disaster happens.

The idea of the scheme is as follows, after the system is started, the SDN controller detects large and small flows and records them, marks the small flow preference router based on the statistical information. When the SDN controller receives the disaster warning signal, the controller informs the GSLB device to stop broadcasting the IP of the disaster-stricken data center, at the same time sends the session information and statistical flow information to other data centers. The normal data center calculates the path for the incoming flow according to the designed LRBFP algorithm.

B. Monitoring Flow Size

Some studies show that the communication traffic inside the data center is divided into a large flow and a small flow. The large flow refers to the data flow whose transmitting data exceeds 10% of the link bandwidth. Through the evaluation of the data center, it is found that 90% of the flows belong to the small flow [12], [13]. In practical applications, large flow requires higher network throughput, and small flow requires lower latency [14].

Because the routing strategy proposed in this paper needs to distinguish between large and small flows, there must be a way to distinguish between large and small flows. Authors in [15] proposed that the most effective way to detect large flow is at the terminal host. On the one hand, the proportion of resources occupied by the switch is less than that of the switch, so as to avoid excessive use of switch-side resources. On the other hand, the flow characteristics depend on the speeds at which the application generates data packets, not the link status of the network. The terminal is more aware of the rate at which the application sends packets. The monitoring principle is to observe the socket cache area of the terminal host. When the size of the data packet with the same characteristics in the cache area exceeds 100kb, it will be marked as a large flow, if 90% of the traffic through the link belongs to small flow, then we mark this link as the small flow preference link. This paper uses the statistical monitoring method on the host side to monitor the large flow. The terminal needs to accurately count and maintain information such as the rate and bytes of the data flow, and then sends the statistical information to the controller through the switch, so as to realize the large and small flow monitoring function of the controller.

We follow the similar approach as in Mahout [14] to use the virtual LAN priority code point (PCP) bits (set as 001 for large flow and 000 for small flow) to indicate the flows.

C. Disaster Warning

With the continuous advancement of scientific technology, such as laser radar, satellite, remote sensing technology, etc., they are used in natural disaster monitoring. By timely monitoring and warning to ensure maximum personal safety and reduce property loss [11].

In our scheme, we have added an interface to the SDN controller of each data center to receive the disaster warning signal and stop advertising the disaster-stricken data center site IP by notifying the GSLB device, and transfer the session information to other data centers. Therefore, subsequent accesses no longer arrive at the disaster-stricken data center. It avoids the server timeout due to disasters and realizes the user unaware.

D. Algorithm Design

When a disaster happens, a large amount of traffic will flow into other data centers in a short period of time, the network controller may not be able to complete the optimization of the routing path within an acceptable time. In order to solve this problem, the usual approach is to develop a parallel route optimization method to reduce the computation time of route

optimization by using the CPU multi-thread parallel computing function. At present, although general-purpose CPU usually has multiple cores, and each core can support multiple threads, due to the limitation of the architecture, there are only dozens to hundreds of threads supported by one CPU, which cannot make a satisfactory parallel gain for parallel routing calculation. In recent years, GPU performance and general computing programming models have been greatly improved. Compared with the CPU, the GPU can support tens of thousands of threads at the same time, and the parallel computing capability is very impressive. Therefore, a GPU-based parallel routing optimization algorithm is more suitable.

Although the GPU can support massive parallel threads, its unique architecture determines its weak ability to handle complex logic. And under the host device programming model, the data must be copied from the main memory to the device memory in each iteration, it will consume extra time. Therefore, an efficient parallel algorithm running on the GPU environment should have the following two characteristics. First, the parallelism of the algorithm should be very high, and the calculation logic of each parallel thread should be simple. Second, the number of iterations of the algorithm should be less.

1) Network Model

This paper models the SDN network as a directed graph $G(V, E)$, where V represents the set of all nodes, and E is the set of all links. $N = |V|$ and $m = |E|$ represents the number of nodes and the number of edges, respectively. For each link $(i, j) \in E$, w_{ij} represents the weight of this link (i, j) (the cost required to transmit one unit of traffic). For each link $(i, j) \in E$, c_{ij} represents the capacity on this link. Suppose D represents the set of business requirements that need to be routed. For business $d \in D$, s_d represents the source node of the business, t_d represents the destination node of the business, and b_{w_d} indicates the required bandwidth of the business d.

2) Problem Formalization

In this section, we model the traffic engineering problem in the network as a mixed-integer programming model. We use network traffic as the input of the problem and find the optimal route to minimize the cost function.

The cost function is usually set to the level of the network congestion assessment. For example, the most commonly used cost function is the maximum link utilization (MLU), which is simply defined as the link utilization of the link with the highest utilization [16], [17]. Others [18], [19] take the sum of the link utilization of all links as the cost function. The logic of the link utilization cost function is:

(1) Low link utilization means low network latency.

(2) Maintaining low link utilization means that more bandwidth is reserved for other services arriving in the future.

However, a large number of experiments based on actual topology show that the link utilization cost function, especially MLU, in the case of a large amount of network congestion, MLU only optimizes the maximum link utilization, but cannot give a feasible (satisfying capacity constraints) solution. So as an alternative, this paper uses routing cost as a cost function. When a disaster happens, a batch of services will arrive within a short period of time. The controller needs to calculate the path that satisfies the link capacity constraints and minimizes the total routing cost. In order to make as many services as possible join the network, we combine the flow characteristics of the data center to give priority to meeting the needs of small flows. Meanwhile, we set the cost of blocked flow to a larger value as follows.

$$\sum_{d \in D} f(d) \tag{1}$$

$$f(d) = \begin{cases} \left(c(p_s) + c(\complement p_d{}^{p_s})\right) \cdot b_{w_d} & joined \\ W \cdot b_{w_d} & blocked \end{cases} \tag{2}$$

Our cost function is the sum of the routing costs of each business, and the routing cost of each business is a branch function. When service can join the network, the routing cost of the service is the product of the service flow size and the path unit cost. For example, in the first branch of $f(d)$, we use p_d to represent the calculated path corresponding to the business d, and p_s represents the links that are marked as small flow preference links, $\complement p_d^{p_s}$ means the absolute complement of p_s in p_d, and $c(p_s)$ means the unit cost of this link set p_s, the unit cost of the link set $c(p_s) = \sum_{(i,j) \in p_d} 10^{xy} w_{ij}$ is the sum of the unit costs of the links it passes through, x indicates whether this set is a small flow preference link set, y indicates whether this service is marked as a large flow service, x, y are 0, 1 integer variables. When the large flow flows through the small flow preference link, we increase the routing cost by a factor of 10 to reduce the impact on the small flow and make the small flow route more quickly, thereby ensuring that as many users as possible are served within limited resources. When the service is blocked, we punish it, such as the second branch of $f(d)$, we set the unit routing cost of the service to a larger value W, which is much greater than all possible path costs of the service. This can punish blocked services, thereby reducing the blocking rate.

In order to make the expression uniform and convenient to express, we first construct the auxiliary edge set E_a. Initially set E_a is empty, then for $\forall v, u \in V, v \neq u$, we add a link (v, u) in E_a and set the capacity of link (v, u) and the cost as ∞ and W respectively. Then, we add the links in set E_a to the original graph $G(V, E)$ to get a new graph $G_b(V, E_b)$, which is $E_b = E \cup E_a$, then graph $G_b(V, E_b)$ has enough capacity to accommodate business needs. If a certain service is routed to the link of data set E_a, it means that this service is blocked. After constructing the auxiliary graph $G_b(V, E_b)$, the cost function of the traffic engineering problem can be expressed as the formula (3).

$$\begin{aligned} z^* = minimize f(d) &= \sum_{d \in D} \left(c(p_s) + c(\complement p_d^{p_s})\right) \cdot b_{w_d} \\ &= \sum_{d \in D} \sum_{(i,j) \in p_d} 10^{xy} w_{ij} \cdot b_{w_d} \end{aligned} \tag{3}$$

$$\sum_{(i,j) \in E_b} x_{ij}^d - \sum_{(j,i) \in E_b} x_{ji}^d = \begin{cases} 1 & if\ i = s_d \\ -1 & if\ i = t_d \\ 0 & otherwise \end{cases} \tag{4}$$

where x_{ij}^d is a 0, 1 integer variable, and $x_{ij}^d = 1$ indicates that the route of the business d has passed through the link (i, j). In order to avoid link congestion, routing needs to meet link capacity constraints in formula (5).

$$\sum_{d \in D} x_{ij}^d \cdot b_{w_d} \leq c_{ij} \ , \quad \forall (i,j) \in E_b \tag{5}$$

In this model, the number of variables increases in multiples with the size of the business volume and network size, so this MILP model is difficult to solve in large-scale situations.

3) Lagrangian Relaxation based Model

In the model (formula (3), formula (4) , formula (5)), the network capacity constraint (formula (5)), links all routing variables together, because the value of these variables must ensure that the traffic occupied of each link is smaller than the capacity of the link. Because of link capacity constraints, the routing of each service becomes not independent of each other. But to take advantage of the parallel nature of the GPU, it is necessary to find the possibility of independent calculation. Therefore, this paper uses the Lagrangian relaxation method to decompose the traffic engineering problem into a batch of routing calculation problems. These routing calculation problems are independent of each other, which are very suitable for parallel computing.

Relax the network capacity constraints in the model (such as formulas (3), (4), (5)) into the objective function then get the following Lagrangian subproblem.

$$\begin{aligned} L(\lambda) = & min \sum_{d \in D} \sum_{(i,j) \in E_b} 10^{xy} w_{ij} \cdot b_{w_d} \cdot x_{ij}^d + \\ & \sum_{(i,j) \in E_b} \lambda_{ij} \left(\sum_{d \in D} b_{w_d} \cdot x_{ij}^d - c_{ij} \right), \end{aligned} \tag{6}$$

where λ_{ij} represents the Lagrange multiplier of link (i, j).

The formula (6) can also be expressed as:

$$\begin{aligned} L(\lambda) = & min \sum_{d \in D} \sum_{(i,j) \in E_b} (10^{xy} w_{ij} + \lambda_{ij}) \cdot b_{w_d} \cdot x_{ij}^d - \\ & \sum_{(i,j) \in E_b} \lambda_{ij} c_{ij} \end{aligned} \tag{7}$$

which is limited by formula (4).

The term $\sum_{(i,j) \in E_b} \lambda_{ij} c_{ij}$ in the objective function of the Lagrange subproblem does not change with the change of the Lagrange multiplier. This paper will discard it as a constant term and will not discuss it. After discarding the term $\sum_{(i,j) \in E_b} \lambda_{ij} c_{ij}$, the objective function of the Lagrangian subproblem only contains the product of the cost parts $10^{xy} w_{ij} + \lambda_{ij}$ and $b_{w_d} \cdot x_{ij}^d$. Note that $\sum_{(i,j) \in E_b} (10^{xy} w_{ij} + \lambda_{ij}) \cdot b_{w_d} \cdot x_{ij}^d$ represents the routing cost of the business d. Therefore, the objective function of the Lagrangian subproblem is to minimize the sum of the routing costs of all services. Observing the constraints of this subproblem, we find that each constraint contains only one variable related to business needs. So this Lagrangian subproblem can be decomposed into a series of independent shortest path problems (each service requirement corresponds to a shortest path problem), but the link cost of these shortest path problems has changed, and the link cost has become related to the Lagrangian multiplier λ, which means that given a Lagrangian multiplier λ, we can consider the Lagrangian subproblem as the shortest of a batch of single services path problem. We can solve this Lagrangian subproblem by calculating a series of shortest paths in parallel.

Because after relaxing the capacity constraint into the cost function, it will not increase the value of the objective function. $L(\lambda)$ becomes the lower bound of the optimal objective function value of the original problem, $z^* \geq L(\lambda)$. In order to get the tightest lower bound value, we have to solve the following optimization problem.

$$L^*(\lambda^*) = maximize_{\lambda} L(\lambda) \tag{8}$$

which is limited by formula (4).

The above optimization problem is also called the dual problem of the original traffic engineering problem (formula (3), formula (4), formula (5)) [20]. Where λ^* represents the optimal Lagrange multiplier. In order to get the optimal multiplier λ^*, we can use the sub-gradient optimization algorithm to solve. When calculating the sub-gradient optimization, the multiplier λ^0 is initialized for the first time, and then update the multiplier by formula (9).

$$\begin{aligned} \lambda_{ij}^{k+1} &= \lambda_{ij}^k + \theta_k g^k \\ &= \lambda_{ij}^k + \theta_k \left[\left(\sum_{d \in D} x_{ij}^d \cdot b_{w_d} - c_{ij} \right) \right]^{+}, \end{aligned} \tag{9}$$

among them, λ_{ij}^k represents the Lagrange multiplier corresponding to the edge (i, j) in iteration k, and g^k is any subgradient of $L(\lambda)$ to λ^k, θ_k indicates the step size of iteration k, and the symbol $[\alpha]^{+}$ indicates the part of the positive sign bit in α , which means $[\alpha]^{+} = max(\alpha, 0)$. It can be seen from the formula (8) that if the sum of the traffic on the link exceeds the capacity on the link (i, j), the Lagrange multiplier λ_{ij}^k of the link (i, j) will increase, which means that some business traffic needs to be removed from the link (i, j). In addition, in order to avoid the link cost of negative weight, when the link capacity is greater than the traffic of it, we do not reduce the λ_{ij}^k of this link (i, j).

Based on the above discussion, we give a parallel traffic engineering algorithm based on the Lagrange multiplier method, which mainly includes the following steps:

1. Initialize link weight for $G(V, E)$
2. Calculate the shortest path of all services, where the path calculation task is assigned to the GPU for parallel calculation.
3. In order to obtain the optimized objective function value of the original problem from the currently calculated path, adjust the path calculated by step 2.
4. Update the link weight. After the update is completed, if the stop condition is not met, return to step 2.

4) GPU based Parallel Routing Calculation

In each iteration, the algorithm calculates the shortest path for each business. However, the logic of the shortest path algorithm is too complex for the GPU, resulting in the inability to fully utilize the GPU's massive parallelism. In order to

improve the degree of parallelism, we need to parallelize the shortest path algorithm.

Authors in [21] proposed a parallel implementation of Dijkstra's shortest path algorithm on the GPU. However, from the analysis of the algorithm structure, Dijkstra's shortest path algorithm is not suitable for the design of parallel algorithms [22], so the implementation of Dijkstra's shortest path algorithm on the GPU cannot get a good acceleration effect. The complexity of the Bellman-Ford [19] algorithm is $(|V| \cdot |E|)$, this algorithm complexity is generally higher than that of Dijkstra's shortest path algorithm. However, because the operation of the Bellman-Ford algorithm is independent of each relaxation edge, thus, the Bellman-Ford algorithm is easier to make parallel on the GPU. In order to get a better acceleration effect, we choose the Bellman-Ford shortest path algorithm for parallel implementation.

The CUDA implementation of the parallel Bellman-Ford algorithm for multiple businesses is shown in the Algorithm 1.

Algorithm 1 Parallel Bellman-Ford Algorithm for Multiple Businesses

Require: Business requirements set D; link set E
Ensure: Set of shortest paths for business requirements P
1: Add the source node of the business to the set S
2: Mark $\leftarrow 1$
3: **while** Mark > 0 **do**
4: Mark $\leftarrow 0$
5: KERNEL_DISTANCE_UPDATE$(S, E, Dist)$
6: **end while**
7: KERNEL_PREDECESSOR_UPDATE$(S, E, Dist, Pre)$
8: Rebuild the shortest path of the business according to the predecessor data Pre, and add the path to the set P

It should be noted that the thread is executed independently on the GPU, there will be synchronization problems when updating the distance mark and the precursor mark of a node. To avoid this synchronization problem, we use two kernels, one is kernel_distance_upadte that is used to update the distance label, the other is kernel_predecessor_update that is used to update the predecessor node.

1: **function** KERNEL_DISTANCE_UPDATE$(S, E, Dist)$
2: $bid \leftarrow blockID$
3: $tid \leftarrow threadID$
4: $s \leftarrow S[tid]$
5: $e \leftarrow E[bid]$
6: **if** $Dist[s][e.tail] + e.weight < Dist[s][e.head]$ **then**
7: $Mark \leftarrow 1$
8: $Dist[s][e.head] \leftarrow Dist[s][e.tail] + e.weight$
9: **end if**
10: **end function**

5) Link Weight Update

In our algorithm, at the $(K+1)$th iteration, the weight of link (i, j) is updated to $w_{ij}^{k} + \lambda_{ij}^{k+1}$, where λ_{ij}^{k+1} is updated to formula (9). In order to ensure convergence, this paper adopts a simple but effective link weight update step size, assuming

1: **function** KERNEL_PREDECESSOR_UPDATE$(S, E, Dist, Pre)$
2: $bid \leftarrow blockID$
3: $tid \leftarrow threadID$
4: $s \leftarrow S[tid]$
5: $e \leftarrow E[bid]$
6: **if** $Dist[s][e.tail] + e.weight = Dist[s][e.head]$ **then**
7: $Pre[s][e.head] = e.tail$
8: **end if**
9: **end function**

that θ_k^{ij} is the link (i, j) at kth iteration. If the step size needs to be updated, then θ_k^{ij} is:

$$\theta_k^{ij} = \frac{1}{|c_{ij} - \sum_{d \in D} x_{ij}^{d} b_{w_d}|} \quad (10)$$

From the formula (10), we can see that if the amount of traffic carried on a link exceeds the capacity of this link, the weight of this link will increase by 1 before the next iteration. For other links whose traffic meets the constraints, their weights will not change. The experimental results show that this coarse-grained update operation greatly reduces the number of convergence iterations of the algorithm, thereby greatly reducing the running time of the algorithm.

The Lagrangian relaxation method decomposes the original problem into individual shortest path problems, which allows the algorithm to be designed in parallel. However, because each subproblem is independent, each problem is greedy when seeking the shortest path, which may cause a large number of services to seize the same batch of links and cause congestion. Once congested, the link weight will increase. And it will cause a large number of business groups to abandon this batch of links to seize other links, so that other links are also congested, forming a vicious circle. Finally, the algorithm converges to the local optimal solution in advance. In addition, in order to pursue fast convergence, we simply increase the weight of each edge that exceeds the capacity constraint by 1. This coarse-grained increase may increase the congestion cycle.

In order to solve the situation where the coarseness of the link increase is too large, this paper uses a random update strategy. For an edge (i, j) whose flow exceeds the capacity constraint, we update the weight with a probability of φ, and the weight increase granularity of each edge is still 1 unit. Our experiment shows that this new strategy can solve the more optimal solution while ensuring the convergence speed of the algorithm.

6) Path Adjustment

Note that under the optimal weight cost (optimal Lagrangian multiplier), the set of links solved by the algorithm is the optimal solution of the Lagrangian dual problem, but it is not necessarily feasible for the original traffic engineering problem. Because each service does not consider the path selected by other services when selecting a path, leads to link conflicts, it is not a feasible solution to the original problem.

To illustrate the business path adjustment algorithm, we introduce some symbols, assuming that P represents the calculated set of paths, where $p_d \in P$ represents the path

of the business d, rp_d represents the available bandwidth of the path p_d, and $rp_d = min\{r_e \mid e \in p_d\}$, where r_e represents the remaining bandwidth of the link e, and D_l represents the remaining service set.

The path adjustment algorithm is shown in Algorithm 2. The main idea of the path adjustment algorithm is to obtain the optimized feasible solution of the original problem by adjusting the path of a small part of the business. The algorithm first sorts the services. On the one hand, to make the objective function smaller, those services with larger traffic requirements are preferentially added to the network. However, if the routing costs of large-traffic services are high, after a long path, a large amount of link capacity resources in the network will be wasted. So the algorithm sorts the current solution and its path according to the value of $\frac{\sqrt{b_{w_d}}}{|p_d|}$, where b_{w_d} represents the amount of traffic required by the business d, and $|p_d|$ represents the cost of the business d through path p_d. After sorting, the algorithm attempts to add services to the network according to the order. If $rp_d \geq b_{w_d}$ indicates that the service can be added to the network, then join the service and update the remaining capacity value of the network link. Otherwise, we add the business to the remaining set D_l. After the loop from line 2 to line 9 ends, we get a remaining network $G'(V', E')$. According to the previous discussion, there may be some equivalent paths in the remaining network, and there are still many available resources in the remaining links, so the algorithm calculates the path for the remaining business D_l in the remaining network (lines 11 to 23). The algorithm sequentially traverses the set D_l to see if it can find a path for the service in the remaining network. First, it removes those links whose residual capacity is less than the service traffic b_{w_d}. Since the remaining links are residual networks, a path with a large hop count may be found. If the hop count is too large, it will occupy too many resources and the optimization goal cannot be achieved. So we restrict the number of hops, if $|p| \leq |p_d|$, then add the service to the network, and update the network link capacity, otherwise we do not add the service to the network. Although this process calculates the path serially, this process is very fast, which is mainly due to the following reasons. First, because the link capacity in the remaining network is generally small, the links that can participate in the calculation are very less. For a remaining business d, before calculating the path, the algorithm will eliminate those links with a residual capacity less than b_{w_d}, so the network topology involved in the calculation is actually very small. Second, the remaining traffic set D_l itself is relatively small. Third, the further the algorithm is executed, the fewer remaining links are available and the smaller the network will become. Therefore, through this path adjustment algorithm, a feasible optimal solution to the original problem can be quickly obtained.

7) Termination Condition

Suppose the optimal solution obtained by this iteration in the first k is B^*, and the optimal solution found in the $(k+1)$th iteration is b^*_{k+1}. If $b^*_{k+1} < B^*$, then $B^* = b^*_{k+1}$. If iterates continuously L times, B^* is still not be updated, the algorithm is determined to converge and the algorithm stops.

Algorithm 2 Path Adjustment Algorithm

Require: Network topology $G(V, E)$; Traffic demand set P; Calculated path set P
Ensure: Adjusted path set AP
1: Sort the business in descending order according to the value of $\frac{\sqrt{b_{w_d}}}{|p_d|}$
2: **for** $d \in D$ **do**
3:   **if** $rp_d \geq b_{w_d}$ **then**
4:     Add path p_d to AP
5:     Update the remaining bandwidth of the link through by path p_d in graph $G(V, E)$
6:   **else**
7:     Add business d to the remaining set D_l
8:   **end if**
9: **end for**
10: $G'(V', E') = G(V, E)$
11: **for** $d \in D_l$ **do**
12:   $G''(V'', E'') = G'(V', E')$
13:   **for** $e \in E'$ **do**
14:     **if** $r_e < b_{w_d}$ **then**
15:       Remove link e from graph $G'(V', E')$
16:     **end if**
17:   **end for**
18:   Calculate the shortest path p for business d in graph $G''(V'', E'')$
19:   **if** $|p| \leq |p_d|$ **then**
20:     Add path p to AP and remove business from set D_l
21:     Update the remaining bandwidth of the link through by path p in graph $G'(V', E')$
22:   **end if**
23: **end for**

III. Experimental Simulation

A. Scheme Test

We use the multi-site data center shown in the following Fig. 1 for testing.

We assume the earthquake is coming at the 10th second and the disaster prevention department can issue an early warning signal 4 seconds ahead. So our scheme can receive the disaster warning signal at the 6th second. And at the 10th second, the data center A suffers an earthquake and the server becomes unavailable. In the experiment, we assume that the expiration time of the DNS cache is 5 seconds, which means the client A needs 5 seconds before requesting the data center B without our scheme.

In order to judge our scheme can forward requests to the normal data center in advance, we simulated 100 requests per second to access the server. Judging the validity by the loss of the request. The experimental results are shown in Fig. 2, we can see that based on our scheme, the request is not significantly lost. But the architecture without the signal warning mechanism, between ten and fifteen seconds, the request is obviously lost until the DNS cache is expired.

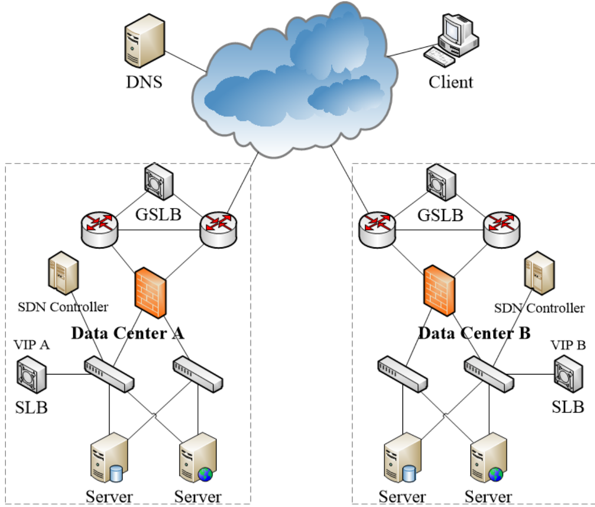


Fig. 1: Multi-site data center network architecture

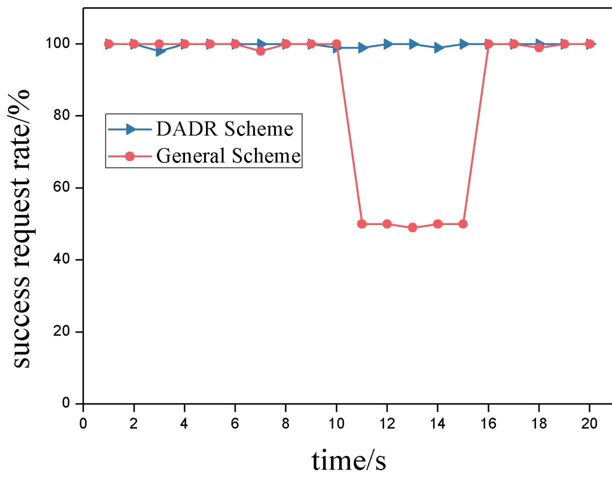


Fig. 2: The total amount of success request

B. Algorithm Test

1) Test setup

In order to show the effectiveness and performance of the algorithm more clearly, this paper makes the simulation by using the Mininet. We build the network topology on the Four-way Server that uses Ubuntu 19.10 system which with the Xeon E5-4669 v3 CPU and Nvidia tesla c2070 GPU, including the OpenvSwitch general host and link supporting OpenFlow protocol. At the same time, an external controller is connected to the Mininet. The controller used in the experimental environment is OpenDaylight (ODL) [23], and the LRBFP algorithm is implemented by CUDA 10.2.89 [24]. Then we add the ODL's function module. The experiment uses the Fat-Tree network topology [25] in Fig. 3, which includes the edge layer switches (S5, S6,..., S39, S40), the aggregation layer switches (S3, S4,..., S37, S38) and the core layer switches (S1,..., S8). Each edge layer switch is connected to two hosts through two ports. Each aggregation layer switch or edge layer switch forms a Pod. Each Pod has an available link and multipath characteristics.

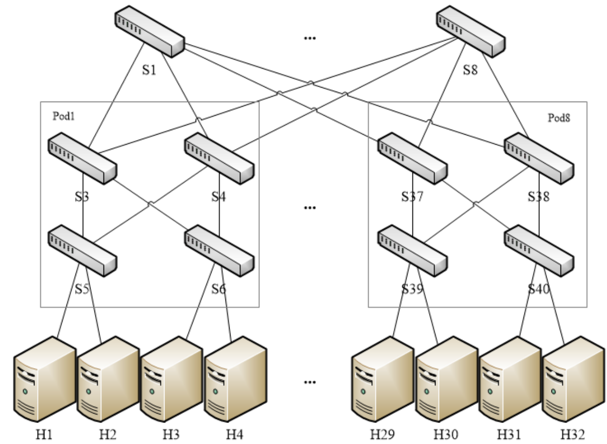


Fig. 3: The Fat-Tree network topology

In the experiment, the data center link bandwidth is $4Gb/s$ and the link delay is $100us$. In the data center server, we choose $H1$, $H5$,...,$H25$, $H29$ as the aggregator, $H2-H4$, $H6-H8$,..., $H26-H28$, $H30-H32$ as the worker. The short flow is simulated by using the query flow of the web search scenario. The client sends a random size data between $1kb$ and $10kb$ request to the random aggregator. The aggregator randomly selects a worker as a receiver. After receiving the request, the worker returns a random size data between $1kb$ and $100kb$ response to the aggregator, and the aggregator returns it to the user. The large flow is simulated by using the download flow of the video viewing scene. The client sends a random size data between $1kb$ and $10kb$ request to the random aggregator, and the aggregator randomly selects a worker as the receiver. After the worker receives it, it returns a random size data between $100kb$ and $10M$ response to the aggregator, which returns to the user. In the simulation we randomly request the query flow and the download flow at a ratio of $1:9$, and send about 3000 requests per second.

After the system has been running for 5 seconds, we doubled the request to 6000 per second, in order to simulate the scenario of a large amount of sudden traffic enters the data center when a disaster happens.

2) Results

In order to verify the ability of our proposed algorithm to deal with a large amount of sudden traffic, the experiment intends to use delay and packet loss rate as comparison parameters. We also compare the proposed LRBFP algorithm with the ECMP algorithm [26] and the Extending Dijkstra's shortest path Algorithm(EDA) [27] instead of the LOBUS, or LBMS, since they are intended to be used in scenarios different from the proposed algorithm.

From Fig. 4 and Fig. 5, we can see that the ECMP algorithm is obviously insufficient in processing capacity when faced with a large amount of sudden traffic, the packet loss rate and delay are high, and the packets loss rate of the proposed LRBFP algorithm is significantly better than the other two algorithms. Although the delay is higher than before, it is within an acceptable range. And compared with the other two algorithms, it is obviously better. The obvious delay increase

of the LRBFP algorithm in 5-8 seconds is due to the algorithm being executed and the system running the default routing strategy. In our proposed scheme, we can actually know the traffic information in advance and conduct calculations. In this experiment, we do not simulate it.

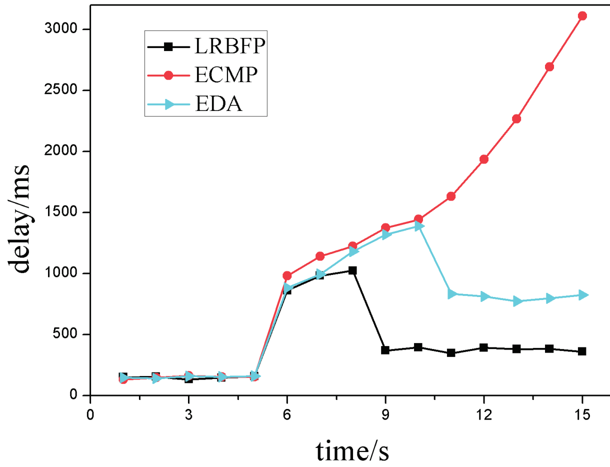


Fig. 4: Delay comparison

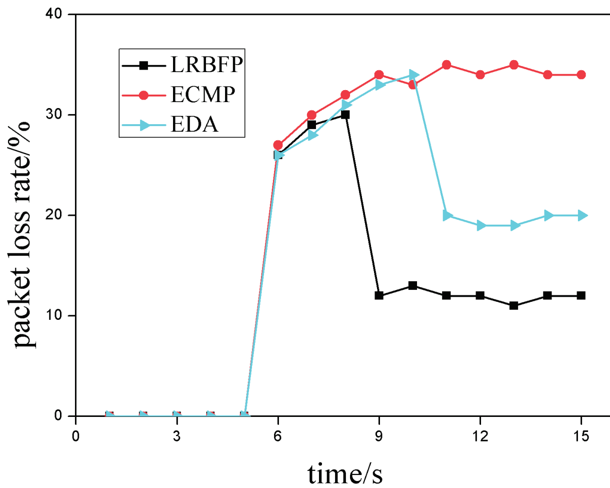


Fig. 5: Packet loss rate comparison

Our experimental results show that when a disaster happens, which results in a large amount of sudden traffic, our LRBFP algorithm can effectively reduce the packet loss rate and delay, and the disaster response-ability is strong.

At the same time, in order to verify that the algorithm can handle more services in a limited time, we conduct an experimental analysis on the running time of the following algorithms, i.e., Serial Dijkstra shortest path Algorithm (SDA) [28], Parallel Dijkstra shortest path Algorithm (PDA) [21] and the proposed LRBFP algorithm. We use the Erdos-Renyi (ER) [29] model to generate the experimental network topology. We set the number of topology nodes to 1000, the link capacity to a random value between 50 and 200, and the link cost to 1. The number of requests gradually increases from 1000 to 10000, and the size of the service is set to a random value between 1 and 100.

In Fig. 6, we can see that with the number of requests increases, the calculation time of the SDA algorithm increases significantly, and the LRBFP algorithm using GPU accelerated parallel computing has a smaller increase. The acceleration advantage of the algorithm becomes larger, which also ensures the algorithm's ability to process massive data at the event of a disaster.

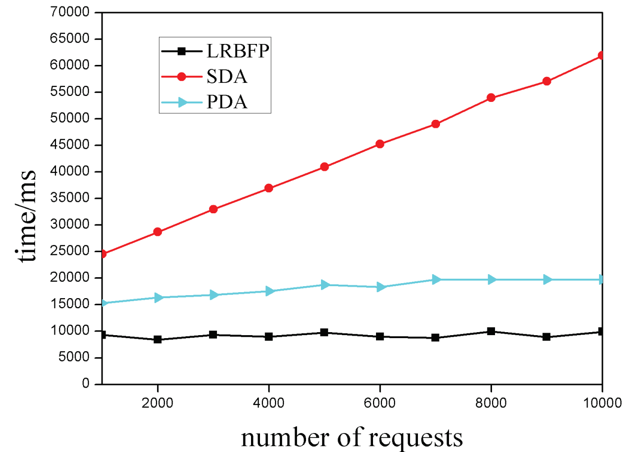


Fig. 6: Calculation time comparison

3) System Load

According to the above experiments, the results show that our algorithm based on GPU parallel computing LRBFP has better performance than traditional CPU algorithms. In order to more clearly show the performance advantages of our LRBFP algorithm, we also detect the load information of GPU and CPU to prove that it won't bring additional overhead to the controller.

As shown in Fig. 7, we count the system load information every second. It can be seen from the statistical results that the average CPU load is maintained at about 40% caused by the simulation program. In Fig. 7, it can be seen that the CPU load has slightly increased after the disaster. This is due to the increased temperature caused by GPU running of the chassis, which slightly affects the CPU performance. In generally, the experimental results show that our algorithm does not bring additional overhead to the system.

IV. Conclusion

With the help of SDN technology, this paper proposed a disaster aware dynamic routing (DADR) scheme. The scheme adopts the SDN centralized control method to receive the disaster warning signal and notify the GSLB device to make corresponding adjustments. At the same time, the SDN controller transfers the session and flow information to other data centers. In this way, it can achieve switching without users' attention and other data centers can calculate the flow path in advance according to the flow information. For the other data centers, combining the characteristics of data center flow, this paper proposed a Lagrangian Relaxation based Bellman-Ford Parallel (LRBFP) algorithm. The main idea is that the controller monitors the data flow, distinguishes between large and small flows, and marks the small flow preference route,

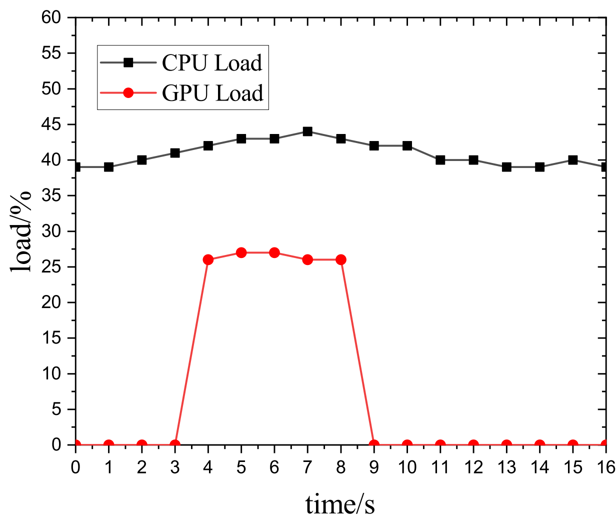


Fig. 7: CPU and GPU average load

models and relaxes the network. We design an algorithm suitable for GPU parallel operation to guide the large flow to avoid such routes for path allocation operations. In this way, under the limited network resources, it can meet the access needs of most users as much as possible. Finally, we validated the proposed DADR scheme and LRBFP algorithm through simulations and made comparisons with the existing traffic scheduling methods. Our scheme can ensure that users are not aware of switching when a disaster occurs. And when a large amount of sudden traffic flows into the data center, our algorithm can provide services for as many users as possible under the case of lower delay.

REFERENCES

[1] T. Adachi, Y. Ishiyama, Y. Asakura, and K. Nakamura, "The restoration of telecom power damages by the great east japan earthquake," in *2011 IEEE 33rd International Telecommunications Energy Conference (INTELEC)*. IEEE, 2011, pp. 1–5.

[2] "Flooding, power outages from hurricane sandy lead to internet, phone service disruptions." https://nypost.com/2012/10/30/flooding-power-outages-from-hurricane-sandy-lead-to-internet-phone-service-disruptions, [Accessed: Dec 2020].

[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems*, vol. 26, no. 2, pp. 1–26, 2008.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007.

[5] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003, pp. 29–43.

[6] R. S. Narayana, M. Raja, R. Mutnuru, and R. Kondamuru, "Systems and methods for gslb mep connection management across multiple core appliances," Apr. 2 2013, uS Patent 8,412,832.

[7] J.-B. Moon and M.-H. Kim, "Dynamic load balancing method based on dns for distributed web systems," in *International Conference on Electronic Commerce and Web Technologies*. Springer, 2005, pp. 238–247.

[8] N. McKeown, "Software-defined networking," *INFOCOM keynote talk*, vol. 17, no. 2, pp. 30–32, 2009.

[9] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-serve: Load-balancing web traffic using openflow," *ACM Sigcomm Demo*, vol. 4, no. 5, 2009.

[10] M. Koerner and O. Kao, "Multiple service load-balancing with openflow," in *13th International Conference on High Performance Switching and Routing*. IEEE, 2012, pp. 210–214.

[11] D. Chen, Z. Liu, L. Wang, M. Dou, J. Chen, and H. Li, "Natural disaster monitoring with wireless sensor networks: a case study of data-intensive applications upon low-cost scalable systems," *Mobile Networks and Applications*, vol. 18, no. 5, pp. 651–663, 2013.

[12] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. ACM, 2009, pp. 202–208.

[13] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.

[14] D. Li, M. Xu, H. Zhao, and X. Fu, "Building mega data center from heterogeneous containers," in *19th IEEE International Conference on Network Protocols*. IEEE, 2011, pp. 256–265.

[15] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *INFOCOM*. IEEE, 2011, pp. 1629–1637.

[16] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: Responsive yet stable traffic engineering," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 253–264, 2005.

[17] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "Cope: traffic engineering in dynamic networks," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, 2006, pp. 99–110.

[18] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing ospf weights," in *INFOCOM*, vol. 2. IEEE, 2000, pp. 519–528.

[19] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," *IEEE/ACM Transactions on networking*, vol. 19, no. 6, pp. 1717–1730, 2011.

[20] R. Ahujia, T. L. Magnanti, and J. B. Orlin, "Network flows: Theory, algorithms and applications," *New Jersey: Rentice-Hall*, 1993.

[21] A. S. Nepomniaschaya and M. A. Dvoskina, "A simple implementation of dijkstra's shortest path algorithm on associative parallel processors," *Fundamenta Informaticae*, vol. 43, no. 1-4, pp. 227–243, 2000.

[22] P. Harish and P. Narayanan, "Accelerating large graph algorithms on the gpu using cuda," in *International conference on high-performance computing*. Springer, 2007, pp. 197–208.

[23] "Opendaylight project proposal [eb/ol]," http://www.opendaylight.org/, [Accessed: May 2020].

[24] "Nvidia. programming guide: Cuda toolkit documentation." https://docs.nvidia.com/cuda/, [Accessed: May 2020].

[25] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," 2008.

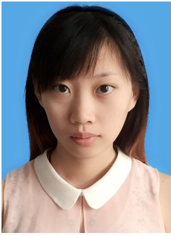
[26] C. Hopps, "Analysis of an equal-cost multi-path algorithm," *RFC 2992, Internet Engineering Task Force*, 2000.

[27] J.-R. Jiang, H.-W. Huang, J.-H. Liao, and S.-Y. Chen, "Extending dijkstra's shortest path algorithm for software defined networking," in *The 16th Asia-Pacific Network Operations and Management Symposium*. IEEE, 2014, pp. 1–4.

[28] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[29] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

Wenhao Zhang received his B.S. degree in Computer Science from Henan University of Science and Technology in 2015, and received his M.S. degree at the school of Systems Information Science, Future University Hakodate, Hokkaido, Japan, in 2020. His research interests include SDN-based Data Center and covert wireless communication.



Xiaochen Li received her B.S. degree in Computer Science from Henan University of Science and Technology in 2015, and received her M.S. degree in Computer Science from Xidian University, Xi'an, China, in 2018, and received her Ph.D. degree from the School of Systems Information Science, Future University Hakodate, Japan, in 2020. She is currently working as an assistant professor at the PLA Strategic Support Force Information Engineering University, China. Her research interests include wireless network security and MANET performance modeling.

Lisheng Ma received his B.S., M.S. and Ph.D degrees from Taiyuan Normal University, China in 2004, from Southeast University, China in 2007 and from Future University Hakodate, Japan in 2017, respectively. He is currently an associate professor of Chuzhou University, China. His research interests include switching networks and data center networks.