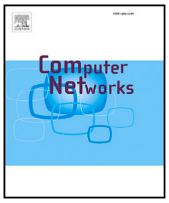
Computer Networks 192 (2021) 108045

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Matchmaker: Maintaining network programmability for Software-Defined WANs under multiple controller failures

Songshi Dou [a], Guochun Miao [b], Zehua Guo [a,*], Chao Yao [c], Weiran Wu [a], Yuanqing Xia [a]

[a] Beijing Institute of Technology, Beijing, 100081, China
[b] Haifeng General Aviation Technology Co., Ltd., Beijing, 100070, China
[c] Shaanxi Normal University, Xi'an, 710119, China

ARTICLE INFO

Keywords:
Software-defined networking
Wide area networks
Control failure
Resiliency
Flow routing
Switch-controller mapping

ABSTRACT

Software-Defined Networking (SDN) provides great opportunities to improve the network performance of Wide Area Networks (WANs). In Software-Defined WANs (SD-WANs), SDN controllers dynamically route flows based on network status by managing underlying switches. However, under controller failures in SD-WANs, existing solutions are unadaptable and thus cannot efficiently map offline switches, which were controlled by failed controllers, to active controllers. Thus, flows, which traverse offline switches, become offline and lose their programmability, which means they cannot be rerouted to accommodate to traffic variation. Consequently, the network programmability degrades. In this paper, we propose Matchmaker, an adaptive solution to recover offline flows under controller failures in SD-WANs. Matchmaker smartly changes the paths of some offline flows to adjust the control cost of offline switches based on given control ability of active controllers. As a result, Matchmaker can efficiently map offline switches to active controllers and increase the number of recovered flows. The simulation results show that Matchmaker outperforms existing solutions by increasing the number of recovered offline flows up to 45% under ATT topology and up to 77% under Belnet topology.

1. Introduction

Software-Defined Networking (SDN) is a promising networking technology to improve the network programmability by enabling flexible flow routing. Wide Area Network (WAN) is one important scenario for SDN, and many technology companies (e.g., Google [1], Microsoft [2]) have deployed SDN in their WANs, known as SD-WANs. An SD-WAN is usually composed of multiple domains, and each domain consists of SDN switches located at different locations and is controlled by one SDN controller [3]. The controller manages underlying switches in its domain and enables dynamic flow routing based on its network status [4]. Flows that traverse SDN switches have the ability to change their paths and are called *programmable flows*.

Essentially, a controller is a network software installed in a physical server or a virtual machine. Due to some unexpected issues (e.g., hardware/software bugs, power failure), the SDN controller may fail, and its controlled switches will be out of control and become *offline*. An *offline switch*[1] can also forward flows based on its configured flow table, but flows, which traverse offline switches, also become *offline* and lose their flow programmability (e.g., the ability to change the paths of flows). As a result, the network variation cannot be accommodated by dynamically routing/rerouting flows, and the network's programmability could degrade. Maintaining network programmability under multiple controller failures is important since multiple controller failures can lead to more significant performance variation than single controller failure due to the severe decrease of network programmability. Controller failure is affected by many factors (e.g., the stability of hardware/software, the reliability of power supply, and the control load of controllers), so several controllers may fail simultaneously or fail successively. Some existing works [5–8] also study the problem of multiple controller failures.

Controller failures could result in serious network performance degradation problem due to the decrease of network programmability in SD-WANs. One possible solution is to remap offline switches to active controllers to let the offline flows be re-controlled by active controllers, which is of great importance to maintaining the network programmability. With the high network programmability, we can dynamically routing/rerouting flows to accommodate the possible network variation, which can significantly improve the network performance.

* Corresponding author.
E-mail address: guolizihao@hotmail.com (Z. Guo).

[1] We use the *failed switch* to denote the switch that is physical failed and cannot forward packet correctly, and the *offline switch* to denote the switch out of control by the active controller.

https://doi.org/10.1016/j.comnet.2021.108045
Received 17 August 2020; Received in revised form 8 March 2021; Accepted 20 March 2021
Available online 27 March 2021

However, the main challenge of this mapping is how to use limited control resource from active controllers to handle the control cost (e.g., the per-flow state pulling [9,10]) for enabling the programmability for offline flows under different controller failures without overloading active controllers. Otherwise, network performance may degrade (e.g., increasing the communication overhead [11,12]) or even cascading controller failure may happen [13].

The state-of-the-art solutions coping with controller failures problem can be categorized into two types, which are the static manner and the dynamic manner. The static ones [5,14–16] select and place the controllers at the proper locations to establish suitable connection between controllers and switches with careful calculation, which could partially mitigate the impact of possible controller failures. However, the static solutions neglect the control cost of different switches and the dynamic change of controllers' control resource. The dynamic ones [17] typically outperform static solutions since controllers and switches' current status are considered in real time. Unfortunately, we cannot get good recovery performance with existing solutions because existing static and dynamic solutions are both unadaptable due to the fixed control cost of switches, which could lead to the underutilization of active controllers. The given control ability of active controllers cannot be efficiently used to recover offline flows, and only a small number of offline flows can be recovered.

In this paper, we propose an adaptive solution named Matchmaker, which recovers offline flows under controller failures in SD-WAN by adaptively adjusting the control cost of offline switches. Typically, the control cost of an SDN switch is used to route flows and collect traffic information, and it is usually proportionally to the number of flows that traverses this switch. Matchmaker smartly changes the paths of some offline flows and thus enables the control cost of offline switches to match the given control ability of active controllers. Consequently, more offline switches can be mapped to active controllers, and the number of recovered flows increases.

The contributions of this paper can be summarized as follows:

- We propose to recover offline flows by jointly changing the paths of offline flows and establishing controller-switch mappings.
- We formulate the joint problem as the Flow Rerouting and Switch Mapping (FRSM) problem, which is a Mixed-Integer Nonlinear Programming (MINLP). To efficiently solve the problem, we reformulate the problem with linear techniques and design an efficient heuristic algorithm named Matchmaker to solve the problem.
- We evaluate the performance of Matchmaker under different controller failure scenarios. The simulation results show that Matchmaker outperforms existing solutions by increasing the number of recovered offline flows up to 45% under ATT topology and up to 77% under Belnet topology, compared with the baseline algorithm

The rest of the paper is organized as follows. In Section 2, we introduce the background and motivation of this paper. Section 3 explains the design consideration of Matchmaker, and Section 4 mathematically formulates our FRSM problem. Section 5 proposes Matchmaker to efficiently solve the problem. We evaluate and analyze the performance of Matchmaker in Section 6. Section 7 introduces related work. Section 8 concludes this paper.

2. Background and motivation

2.1. Background

2.1.1. SD-WAN

An SD-WAN is usually composed of multiple network domains. Each domain contains several SDN switches and has a master SDN controller to quickly response to the requests from the SDN switches within the domain. Fig. 1 shows an example of the SD-WAN. In this figure, an SD-WAN is composed of three domains, and each domain is managed by

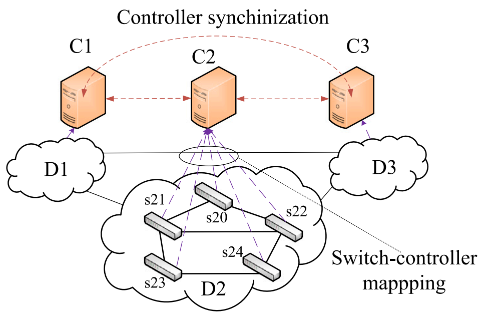
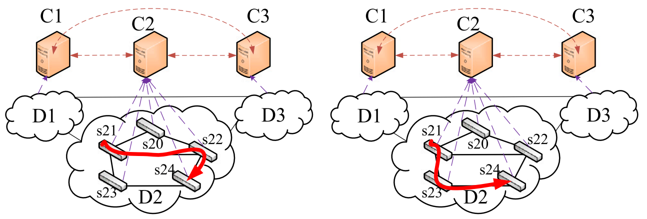


Fig. 1. A simple example of the SD-WAN.



(a) Flow f's next hop in $s21$ is $s22$.

(b) Flow f's next hop in $s21$ is $s23$.

Fig. 2. Changing programmable flow f's path at switch s_{21}.

one master controller. We mainly focus on domain D_2, and this example does not show details of the other two domains. Controller C_2 is the master controller of domain D_2 and controls five SDN switches s_{20}–s_{24} in D_2. The three controllers maintain the consistent network information via controller synchronization. Thus, all the three controllers have the consistent information for D_2.

2.1.2. Programmable flows

SDN enables flexible control on flows to improve the WAN's network programmability by dynamically establishing and changing forwarding paths for individual flows. If a flow traverses an SDN switch, which is controlled by an SDN controller, the controller can change this flow's path, and this flow becomes a programmable flow. Fig. 2 illustrates an example of changing the path of a programmable flow. In this example, flow f is from switch s_{21} to switch s_{24}. In Fig. 2(a), f is forwarded on path $s_{21} \rightarrow s_{22} \rightarrow s_{24}$. In Fig. 2(b), the controller changes f's path to $s_{21} \rightarrow s_{23} \rightarrow s_{24}$ by installing, deleting, and updating flow entries in the related switches.

2.1.3. Resilient network control

The programmability of a flow is realized by the SDN controller to manage this flow's path in underlying switches. If one controller fails, its controlled switches will be out of control and become *offline*. To provide resilient network control, one switch usually connects to several backup controllers. Once the master controller fails, one of the backup controllers will become the new master controller for the domain [18]. The controllers from different domains are physically distributed but maintain a consistent network view by state synchronization among them. Fig. 3 shows an example of master controller changing under a controller failure. In Fig. 3(a), switch s_{20} connects to its master controller C_2 and backup controllers C_1 and C_3, and the three controllers synchronize with others. In Fig. 3(b), when C_2 fails, s_{20} identifies its connection to C_2 become inactive and then sends a master controller selection request to C_1, C_2, and C_3. C_1 replies an accepted message to switch s_{20}'s request. After the connection establishment between C_1 and s_{20}, C_1 becomes the new master controller of s_{20}. This dynamic controller role configuration feature offers resilient network control.

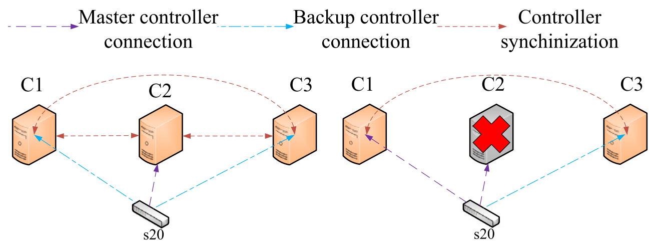

Master controller connection
Backup controller connection
Controller synchinization
C1
C2
C3
s20
C1
C2
C3
s20
(a) Switch s20 connects one master controller C2 and two backup controllers C1 and C3.
(b) Under C2 failure, C1 becomes the new master controller of switch s20.

Fig. 3. Resilient network control.

2.1.4. *Maintaining network programmability*

With the high network programmability, we can dynamically route/reroute flows to accommodate the possible network variation and improve network performance. However, when the controller fails, its controlled switches will be out of control and become offline. The flows, which traverse offline switches, also become offline and lose their flow programmability. It is the controller failures that decreases the network programmability. As a result, the network variation cannot be accommodated by dynamically routing/rerouting flows due to the low network programmability, which proves that it is of vital importance maintaining network programmability under controller failures. To maintain network programmability of SD-WANs under unexpected controller failures, recovering offline flows is vital. Recovering offline flows is to remap offline switches to active controllers. By mapping an offline switch to an active controller, all flows that traverse this switch are controlled by this controller and become programmable.

2.2. *Existing works and the limitation*

2.2.1. *State-of-the-art solutions*

The state-of-the-art solutions recover the path programmability by mapping switches to controllers either in a static manner or a dynamic manner. The static solutions optimize the network deployment by carefully selecting the placement of controllers and the connection between controllers and switches to alleviate the impact of the potential controller failure. Tanha et al. [5] define a resilient controller placement problem, which can improve the reliability and performance of SDN in WANs. Killi et al. [14] propose a generalized model that take the average latency from switches to controllers into account, and extended it for multiple controller failures.

The dynamic solution considers the network variation in real time to remap offline switches to active controllers. Guo et al. [17] propose RetroFlow, which can intelligently configure a set of selected offline switches working under the legacy routing mode to relieve active controllers' control cost of remapping the selected offline switches while maintaining the network programmability.

But both of the static solutions and dynamic solutions are unadaptable solutions which have obvious defects. We will further analyze the problems that unadaptable solutions suffer from.

2.2.2. *Limitation of existing unadaptable solutions*

Establishing mappings is a complicated task. First, the number of offline flows recovered to be programmable should be maximized. Second, the normal operations of active controllers should not be interrupted when recovering offline flows. Thus, an offline switch can be mapped to one active controller only when the controller has enough control ability to handle the switch's control cost, which refers to the load of controlling offline flows in offline switches. Existing solutions are unadaptable and have the following limitations, which could affect the recovery of offline flows:

- **Low utilization efficiency of active controllers**: The unadaptable solutions have a low efficiency utilizing the processing ability of active controllers. The control cost of an SDN switch is used to route flows and collect traffic information, and it is usually proportionally to the number of flows that traverses this switch. Following the given routing decided by the failed controller, the fixed control cost of offline switches do not always match the given control ability of active controllers. Hence, some offline switches cannot be mapped to active controllers. Even though the active controllers are fully utilized, only a small number of offline flows are recovered. That is because the switches are different and we need to carefully choose the proper switches to remap instead of treating different switches as the same.
- **Recovering a small number of offline flows**: Since some offline switches cannot be mapped to active controllers, many offline flows cannot be recovered. Thus, some recovered flows can have high programmability with multiple rerouting paths while others are not recovered and cannot be rerouted at all.

2.3. *Example*

Fig. 4 shows an example to illustrate the above two limitations. In this example, C_2 is the master controller of five switches in D_2, and C_1 and C_3 are backup controllers of them. In Fig. 4(a), controller C_2 fails, and the control of the three switches in D_2 must be handed over to two active controllers C_1 and C_3 to control flows in D_2. Fig. 4(b) illustrates the flows in D_2 and their previous paths decided by C_2. Based on the previous routing, s_{20}–s_{24}'s control cost are all two flows, respectively. The control cost of a switch equals the total number of flows in the switch's flow table. We measure a controller's control ability as the number of flows that the controller can normally process for its operations (e.g., pulling network status from its controlled switches without introducing extra queueing delay). In this example, without interrupting a controller's normal operations, both C_1 and C_3 are only able to control five flows, respectively, and the total control ability of two controllers is to control ten flows. With existing unadaptable solutions to recover offline flows, switches s_{21} and s_{23} are mapped to controller C_1, and switches s_{20} and s_{22} are mapped to controller C_3. s_{24} is not mapped. Two issues are raised:

1. Low utilization efficiency of active controllers: In Fig. 4(c), controller C_1 controls four flows (i.e., f_1 and f_3 from s_{21}, f_2 and f_4 from s_{23}), and controller C_3 controls three flows (i.e., f_1 and f_3 from s_{20}, f_1 and f_2 from s_{22}). Both the two controllers has the remaining control ability for one flow. However, there is no way to remap s_{24} to two controllers at the same time. Thus, the rest control ability of two flows are wasted.
2. Recovering a small number of offline flows: In Fig. 4(c), s_{20}–s_{23} are remapped, but switch s_{24} remains unremapped. A flow can be programmable if all switches on its path are mapped to active controllers. Thus, flows f_2 and f_4 are still offline flows due to the failed remapping of switch s_{24}.

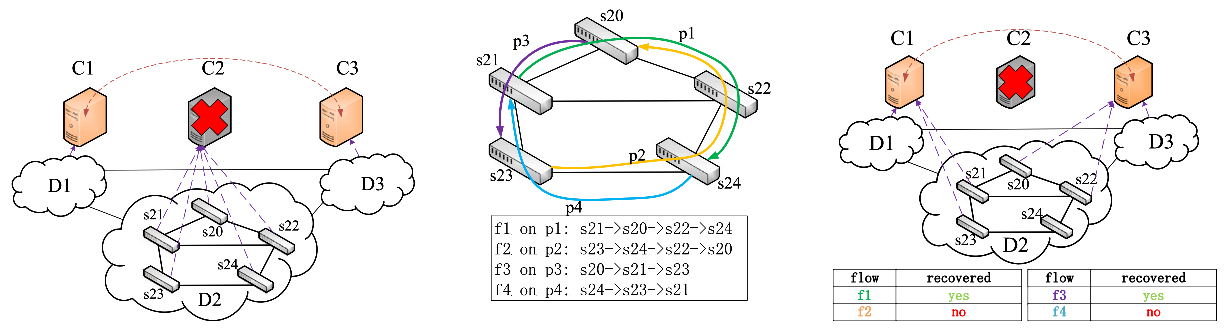
The above examples show that existing unadaptable solutions cannot well recover offline flows under controller failures.

3. Design considerations

In this section, we analyze the root cause of existing unadaptable solution and introduce key design considerations of our solution.

3.1. *Root cause of existing unadaptable solutions*

The root cause of existing solutions mainly comes from establishing switch-controller mappings based on the fixed control cost of switches. For offline flows, their previous routing is decided the failed controller, which has enough control ability to control all flows in them and



f1 on p1: s21->s20->s22->s24
f2 on p2: s23->s24->s22->s20
f3 on p3: s20->s21->s23
f4 on p4: s24->s23->s21

flow	recovered
f1	yes
f2	no

flow	recovered
f3	yes
f4	no

(a) When controller $C2$ fails, switches in $D2$ should be controlled by active controllers $C1$ and $C3$.

(b) Flows $f1$-$f4$ and their paths in domain $D2$.

(c) Offline switches s_{20}-s_{24} are remapped, but switch s_{24} remains unremapped.

Fig. 4. A motivation example of existing unadaptable solutions that establish controller-switch mappings under one controller failure.

provides programmable paths for these flows. However, when the controller fails, the rest active controllers may not have enough control ability to fully control offline switches, and thus cannot offer the same programmable paths for offline flows. As a result, the fixed control cost of switches based on the previous routing prevents efficiently establishing switch-controller mappings. To fully use the control ability of active controllers to recover offline flows, a promising solution is to adaptively adjust the paths of offline flows to let the control cost of offline switches accommodate to the control ability of active controllers. Thus, more offline switches can be mapped to active controllers, and the number of recovered flows can increase.

3.2. Design considerations

In our solution design, changing the path of flows should aim at maximizing the number of recovered flows. We use Fig. 5 as an example to show how this factor affects the performance of our solution under the controller failure in Fig. 4. In Fig. 5(a), flow f_3's path changes from its original path $p_3 : s_{20} \to s_{21} \to s_{23}$ to path $p_3' : s_{20} \to s_{22} \to s_{24} \to s_{23}$. Following this path change, in Fig. 5(b), s_{21} and s_{23} are mapped to C_1, and s_{20} and s_{22} are mapped to C_3. With this solution, s_{24} cannot be remapped to active controllers even though C_1 still has the control ability of two flows. Controller C_1 controls three flows (i.e., f_1 from s_{21}, f_2 and f_4 from s_{23}), and controller C_3 controls five flows (i.e., f_1 and f_3 from s_{20}, f_1, f_2 and f_3 from s_{22}). With this solution, s_{24} cannot be remapped to active controllers because there are three flows traverse s_{24}, but C_1 only has the control ability of two flows, and C_3 runs out of its resources. As a result, only flow f_1 is recovered, and the rest three flows not recovered due to the failed remapping of switch s_{24}. Note that this result is worse the solution in Fig. 4. In other words, inappropriate path change may further degrade flow recovery performance.

Fig. 6 shows the result of Matchmaker, which considers the number of recovered programmable flows. The mapping between offline switches and active controller, and the changed path are decided by solving the FRSM problem, which will be detailed in the following section. In Fig. 6(a), flow f_2's path changes from its original path $p_2 : s_{23} \to s_{24} \to s_{22} \to s_{20}$ to path $p_2' : s_{23} \to s_{21} \to s_{22} \to s_{20}$. Following this path change, in Fig. 6(b), s_{21} and s_{23} are mapped to C_1, and s_{20}, s_{22}, and s_{24} are mapped to C_3. This solution makes full use of control ability of four active controllers and recovers all the four offline flows. Thus, considering the number of recovered programmable flows gives us more opportunity to properly change the path of flows and establish mappings.

4. Problem formulation

In this section, we formulate the FRSM problem and reformulate it with linear techniques to efficiently solve it.

4.1. System description

Typically, an SD-WAN consists of H controllers at H locations, and each controller controls a domain of switches. Controllers $C_{M+1}, \dots, C_H$ fail, and they control N switches in total. The set of active controllers is $\mathcal{C} = \{C_1, \dots, C_j, \dots, C_M\}$, and the set of switches controlled by the failed controllers are $\mathcal{S} = \{s_1, \dots, s_i, \dots, s_N\}$. We try to map switches in $\mathcal{S}$ to controllers in $\mathcal{C}$. The set of flows from $\mathcal{S}$ is $F = \{f_1, \dots, f_l, \dots, f_L\}$. For flow $f_l \in \mathcal{F}$, its path-set consists of K paths and is denoted as $P_l = \{p_l^1, \dots, p_l^k, \dots, p_l^K\}$. The set of paths for $\mathcal{F}$ is $\mathcal{P} = \{P_1, P_2, \dots, P_l, \dots, P_L\}$. If path p_l^k traverses switch s_i, we have $\beta_i^{lk} = 1$, otherwise $\beta_i^{lk} = 0$. If f_l selects path p_l^k, we have $x_l^k = 1$, otherwise $x_l^k = 0$. If flow f_l is recovered to be programmable, we have $y^l = 1$; otherwise $y^l = 0$. We use $z_{ij} = 1$ to denote that switch $s_i \in \mathcal{S}$ is mapped to controller $C_j \in \mathcal{C}$; otherwise $z_{ij} = 0$.

4.2. Constraints

4.2.1. Switch-controller mapping constraint

Each switch can be only mapped to one controller. Thus, we have

$$\sum_{j=1}^{M} z_{ij} = 1, \forall i. \tag{1}$$

4.2.2. Flow-path selecting constraint

Each flow can be forwarded on one path or not be forwarded. Thus, we have

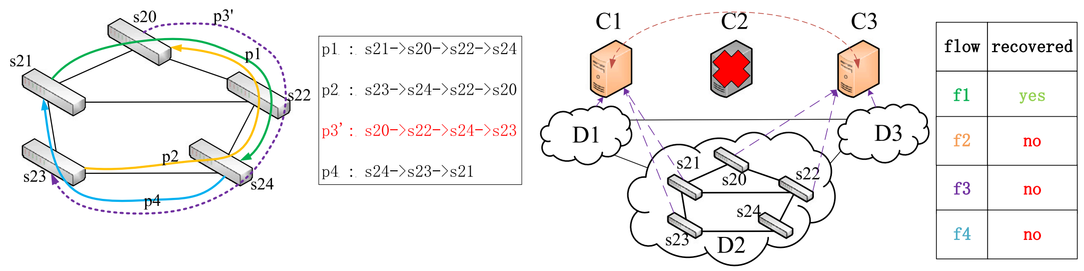
$$\sum_{k=1}^{K} x_l^k \le 1, \forall l. \tag{2}$$

4.2.3. Control ability constraint

If some controllers fail, active controllers should try their best to control the offline switches previously controlled by failed controllers without interrupting their normal operations. The control cost of a switch equals the total number of flows in the switch's flow table. We measure a controller's control ability as the number of flows that the controller can normally process for its operations (e.g., pulling network status from its controlled switches without introducing extra queueing delay). The control ability of a controller should not exceed its received control cost of switches. It can be written as follows:

$$\sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{l=1}^{L} x_l^k * \beta_i^{lk} * z_{ij} \le A_j^{rest}, \forall j, \tag{3}$$

where A_j^{rest} denotes the available resource of controller C_j.



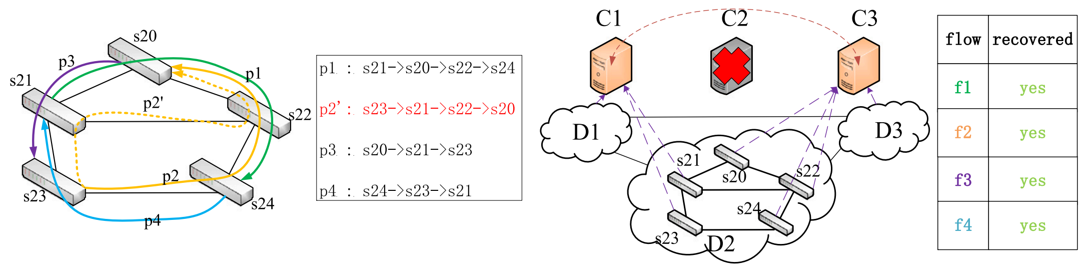
p1 : s21->s20->s22->s24
p2 : s23->s24->s22->s20
p3' : s20->s22->s24->s23
p4 : s24->s23->s21

flow	recovered
f1	yes
f2	no
f3	no
f4	no

(a) Flows $f1$, $f2$, and $f4$ follow their previous paths, but flow $f3$ changes its path to $p3'$.

(b) Only flow $f1$ is recovered under the controller-switch mappings based on the path change in (a).

Fig. 5. The result of a solution that does not consider offline flow recovery efficiency.



p1 : s21->s20->s22->s24
p2' : s23->s21->s22->s20
p3 : s20->s21->s23
p4 : s24->s23->s21

flow	recovered
f1	yes
f2	yes
f3	yes
f4	yes

(a) Flows $f1$, $f3$, and $f4$ follow their previous paths, but flow $f2$ changes its path to $p2'$.

(b) Flow $f1$-$f4$ are all recovered under the controller-switch mappings based on the path change in (a).

Fig. 6. The result of Matchmaker.

4.2.4. Programmable flow constraint

A flow is recovered to be programmable if at least one switch on this flow's path is recovered. The flow f_l's programmability can be expressed as follows:

$$y^l \le \sum_{k=1}^{K}\sum_{j=1}^{M}\sum_{i=1}^{N} x_l^k * \beta_i^{lk} * z_{ij}, \forall l. \tag{4}$$

In the above inequality, the equal sign works when flow f_l is recovered only from one offline switch.

4.3. Objective function

Our objective is to recover as many flows to be programmable as possible. The number of flows that recovered to be programmable is formulated as follows:

$$obj = \sum_{l=1}^{L} y^l.$$

4.4. Problem formulation

The goal of our problem is to rationally select paths in $\mathcal{P}$ for rerouting flows in $\mathcal{F}$ and mapping switches in $\mathcal{S}$ to active controllers in C to maximize the number of recovered flows. We formulate the FRSM problem as follows:

$$\begin{aligned} \max_{x,y,z} \quad & \sum_{l=1}^{L} y^l \\ \text{s.t.} \quad & (1), (2), (3), (4), \\ & x_l^k, y^l, z_{ij} \in \{0,1\}, \forall i,j,l,k, \end{aligned} \tag{P}$$

where $\{x_l^k\}$, $\{y^l\}$, and $\{z_{ij}\}$ are design variables, $\{\beta_i^{lk}\}$ and $\left\{A_j^{\text{rest}}\right\}$ are constants.

4.5. Problem reformulation

In the FRSM problem, the objective function is linear, variables are binary integers, and constraints are nonlinear. Thus, this problem is a Mixed-Integer Nonlinear Programming (MINLP). One complexity of the FRSM problem comes from Eqs. (3) and (4) since two binary variables are multiplied in the two equations. To efficiently solve the problem, we reformulate the problem to an Integer Programming (IP) by equivalently linearizing the bilinear terms of binary variables $x_l^k * \beta_i^{lk} * z_{ij}$ in Eqs. (3) and (4). Specifically, we can replace the bilinear term $x_l^k * \beta_i^{lk} * z_{ij}$ by introducing an auxiliary binary variable ω_{ij}^{lk} and add the following linear constraints:

$$z_{ij} * \beta_i^{lk} \ge \omega_{ij}^{lk}, \forall i,j,l,k, \tag{5}$$

$$x_l^k * \beta_i^{lk} \ge \omega_{ij}^{lk}, \forall i,j,l,k, \tag{6}$$

$$(x_l^k * \beta_i^{lk} + z_{ij} * \beta_i^{lk} - 1) \le \omega_{ij}^{lk}, \forall i,j,l,k. \tag{7}$$

Thus, Eqs. (3) and (4) can be respectively reformulated as follows:

$$\sum_{i=1}^{N}\sum_{k=1}^{K}\sum_{l=1}^{L} \omega_{ij}^{lk} \le A_j^{rest}, \forall j, \tag{8}$$

$$y^l \le \sum_{k=1}^{K}\sum_{j=1}^{M}\sum_{i=1}^{N} \omega_{ij}^{lk}, \forall l. \tag{9}$$

Therefore, FRSM problem can be reformulated as follows:

$$\begin{aligned} \max_{x,y,z,\omega} \quad & \sum_{l=1}^{L} y^l \\ \text{s.t.} \quad & (1), (2), (5), (6), (7), (8), (9), \\ & x_l^k, y^l, z_{ij}, \omega_{ij}^{lk} \in \{0,1\}, \forall i,j,l,k. \end{aligned} \tag{P'}$$

Algorithm 1 Matchmaker

```
Input: $\mathcal{A}$, $\lambda$, $\mathcal{P}$;
Output: $\mathcal{X}$, $\mathcal{Z}$;

1: RoutedFlows = $\emptyset$, MappedSwitches = $\emptyset$, SwitchCosts = $\emptyset$, $\mathcal{A}^* = \mathcal{A}$;
2: Generate $TEST\_Z = \{(i,j), i \in [1,N], j \in [1,M]\}$ and $TEST\_X = \{(l,k), l \in [1,L], k \in [1,K]\}$ by solving the relaxed linear programming of problem (P′) and sorting the results of $z$ and $x$ in the descending order;
3: // Step 1: pre-assign a path for each flow to obtain the control cost of switches;
4: for $(l_0, k_0) \in TEST\_X$ do
5:     if $l_0 \in$ RoutedFlows then
6:         continue; // one flow can select only one path;
7:     end if
8:     RoutedFlows ← RoutedFlows $\cup l_0$;
9:     $\mathcal{X}^* \leftarrow \mathcal{X}^* \cup (l_0, k_0)$
10: end for
11: Use $\mathcal{X}^*$ and $\mathcal{P}$ to generate SwitchCosts = $\{g_i, i \in [1,N]\}$, where $g_i$ denotes the control cost of switch $s_i$.
12: // Step 2: establish controller-switch mappings based on the generated control cost of switches;
13: for $(i_0, j_0) \in TEST\_Z$ do
14:     if $i_0 \in$ MappedSwitches or $A^*_{j_0} \le \lambda * g_{i_0}$ then
15:         continue; // one switch can be remapped to only one controller, and the controller is not overloaded.
16:     end if
17:     $A^*_{j_0} = A^*_{j_0} - \lambda * g_{i_0}$, $\mathcal{Z} \leftarrow \mathcal{Z} \cup (i_0, j_0)$,
18:     MappedSwitches ← MappedSwitches $\cup i_0$;
19: end for
20: // Step 3: assign a path for each flow based on established controller-switch mappings;
21: for $(l_0, k_0) \in \mathcal{X}^*$ do
22:     $\mathcal{A}^* = \mathcal{A}$;
23:     for $i_0 \in p^{k_0}_{l_0}$ do
24:         use $i_0$ to find $j_0$ from $\mathcal{Z}$, $A^*_{j_0} = A^*_{j_0} - 1$;
25:         if $A^*_{j_0} < 0$ then
26:             go to line 22;
27:         end if
28:     end for
29:     $\mathcal{A} = \mathcal{A}^*$, $\mathcal{X} \leftarrow \mathcal{X} \cup (l_0, k_0)$;
30: end for
31: return $\mathcal{X}, \mathcal{Z}$;
```

5. Solution

5.1. The inner working mechanisms of Matchmaker

Fig. 7 shows the inner working mechanisms of Matchmaker. At the beginning, controllers are at the initial state. Once the controller failure is detected, Matchmaker is used to smartly reroute flows and reassign offline switches to active controllers. Then at the recovery state, the situations of controller recovery can be divided into three scenarios. In scenario one, all of the failed controllers recover from the initial failure. Under this scenario, Matchmaker reassigns all switches to the proper controllers following the assignment before the initial failure. In scenario two, only part of the failed controllers recovers from the initial failure. In this case, Matchmaker will identify the controllers, which are still in trouble, and recalculate the proper mapping relationship between switches and controllers based on the current failure case. In scenario three, none of the controllers are recovered. Thus, the state remains unchanged until the recovery of controllers is completed.

5.2. Heuristic solution

Typically, as a Mixed-Integer Nonlinear Programming (MINLP), the proposed FRSM problem is NP-hard with high complexity. The typical solution of the above FRSM problem is to get its optimal result with IP optimization solvers. However, as the network size increases, the solution space could increase significantly, and finding a feasible solution may need a very long time or perhaps is impossible. We achieve the trade-off between the performance and time complexity by solving the problem with the proposed heuristic algorithm Matchmaker.

The key idea of Matchmaker is to test and decide the proper switch-controller mapping relationship and flow-path selection relationship by following the result probability, which can be received by solving the relaxed linear programming of problem (P′). Among the two relationships, the switch-controller mapping relationship has the first priority since we aim to map all offline switches to active controllers. However, simply following the results of mapping variables z do not work well because z depends on path selection variables x, which are also decimal variables, and we cannot use z and x at the same time. To efficiently solve the problem, our proposed Matchmaker works in three steps. In Step 1, we follow path selection variables x to pre-assign a path for each flow and deploy flows' paths into switches to estimate the control cost of switches. In Step 2, we establish controller-switch mappings based on the generated control cost of switches. To ensure that all switches can be mapped to controllers, we use a constant $\lambda \in [0,1]$ to adjust the control cost of switches. In Step 3, given established controller-switch mappings, we finally assign a path for each flow and deploy the path into the related switches without violating the controller's control ability. If the violation occurs, the flow will not be recovered.

Details are summarized in Algorithm 1, and notations used in the algorithm are listed in Table 1. At the beginning of the algorithm, in line 1, we initialize RoutedFlows, MappedSwitches, SwitchCosts, $\mathcal{A}^*$. In line 2, we generate $TEST\_Z$ and $TEST\_X$. The sorting operation helps us to test the mappings based on their probabilities.

Step 1 includes lines 3–10. We test all possible flow-path selection in $TEST\_X$. Lines 5–8 ensure that one flow can select only one path. In line 9, the tested flow and its selected path are stored in $\mathcal{X}^*$. In line 11, we use $\mathcal{X}^*$ and $\mathcal{P}$ to generate SwitchCosts.

Step 2 includes lines 12–19 to test the switch-controller mappings. Lines 14–16 ensure that one switch can be remapped to only one controller, and the controller is not overloaded. If one mapping passes the test, in lines 17 and 18, we update $\mathcal{Z}$ to incorporate this mapping and update the controller ability accordingly.

Step 3 includes lines 20–30 to decide the path of each flow. Lines 22–28 test if deploying flow l_0's path k_0 into related switches violating the controller's control ability. If not, the path is selected for this flow and deployed, and this flow is recovered, as line 29 shows. Otherwise, the next flow and its path will be tested. In line 31, after all flows are tested, the algorithm returns the result and stops.

5.3. Complexity analysis

The time complexity of Matchmaker is $\mathcal{O}(N^2)$, where N denotes the number of switches in the network. Matchmaker consists of three steps. In the first step, there are L iterations, where L is equal to N^2 and denotes the number of flows in the network. In each iteration, the algorithm pre-assigns a path for each flow, which takes a complexity of $\mathcal{O}(1)$ because each flow can only selects one path. Thus, the time complexity for the first step is $\mathcal{O}(N^2)$. In the second step, there are at most $M * N$ iterations, where $M(M < N)$ denotes the number of controllers in the network. In each iteration, the algorithm tries to establish controller-switch mapping based on the generated control cost of switches. The second step takes a time complexity of $\mathcal{O}(M * N)$. As for the third step, there are L iterations. In each iteration, the algorithm ensures that deploying the path of a flow into the network (i.e., the switches on the path) does not overload the controller. According to

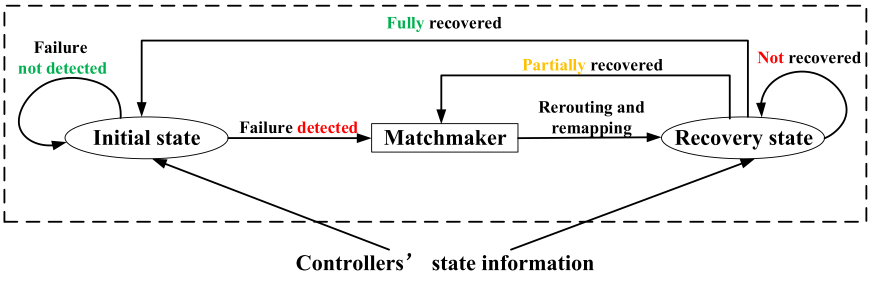


Fig. 7. The inner working mechanisms of Matchmaker.

Table 1
Notations.

Notation	Meaning
$\mathcal{A}$	The set of the control ability of active controllers, $\mathcal{A} = \{A_j^{rest}, j \in [1, M]\}$
$\mathcal{P}$	The set of paths for flows $\in \mathcal{F}$, $\mathcal{P} = \{P_1, \dots, P_l, \dots, P_L\}$, where $P_l = \{p_l^1, \dots, p_l^k, \dots, p_l^K\}$, $l \in [1, L]$.
$TEST\_Z$	The set of testing switch-controller mapping
$TEST\_X$	The set of testing flow-path selection
λ	A constant $\in [0, 1]$

the "small-world" property [19], any two nodes in most real network topologies are reachable in less than 6 hops, which means that it takes a time complexity of $\mathcal{O}(6)$ at most in each iteration. Thus, the time complexity for the third step is $\mathcal{O}(N^2)$. Therefore, the time complexity of Matchmakeris to cumulate the time complexity of three steps together and is $\mathcal{O}(N^2)$.

5.4. Application scenarios

Dealing with the controller failure problem when traffic matrix is known or not are two separate issues. If the traffic matrix is known, we can design a new objective to improve network performance (e.g., load balancing performance) based on given traffic matrix under controller failures. However, traffic matrix provided by service providers is not easily accessible in a timely manner. Thus, a solution to efficiently cope with the controller failure problem under both scenarios (i.e., when traffic matrix is known or not) is needed. Typically, when traffic matrix is not known, increasing the number of routes used for routing flows can potentially improve network performance. In SDN, the number of routes denotes the network programmability. Thus, maintaining network programmability is an effective way to maintain network performance under the controller failure, and the design of Matchmaker follows this idea. In future, we will further consider the controller failure problem when the traffic matrix is known.

6. Simulation

6.1. Simulation setup

We use two typical backbone topologies named ATT and Belnet from Topology Zoo [20] to evaluate the performance of Matchmaker. The ATT topology is a national topology of US and consists of 25 nodes and 112 links. The Belnet topology is a national topology of Belgium and consists of 20 nodes and 62 links. In ATT and Belnet topologies, each node has a unique ID and its latitude and longitude. In backbone networks, each node represents a city or an area, and every pair of two nodes always has a flow. Thus, in our simulation, we only use one traffic scenario, where each node is an SDN switch, and any two nodes have a traffic flow forwarded on the shortest path. In this way, we only need one simulation for this unique traffic scenario under one controller failure case. Flows for each simulation process remain unchanged, and the simulation results will not have any difference if we repeat the simulation. The control plane of ATT consists of six controllers, and of Belnet consists of five controllers The processing ability of each controller is 500. Tables 2 and 3 show the default relationship of controllers, switches, and the number of flows in the switches under ATT and Belnet.

6.2. Comparison algorithms

1. SwitchOnly: during controller failures, offline flows are recovered by mapping offline switches to active controllers as many as possible.
2. Optimal: it is the optimal solution of the problem that maximize the total number of recovered offline flows. We solve the problem using GUROBI solver [21].
3. Matchmaker: this algorithm is shown in Algorithm 1.
4. GWBM-RS [8]: this is a greedy algorithm to assign master and slave controller to switches against multiple controller failures.

6.3. Simulation results

We compare the performance of our solution with other algorithms under one controller failure, two controller failures and three controller failures. Following our objective, we use the following metrics: (1) the percentage of recovered offline flows, (2) the processing load of active controllers, and (3) the percentage of recovered offline switches.

6.3.1. Under ATT topology

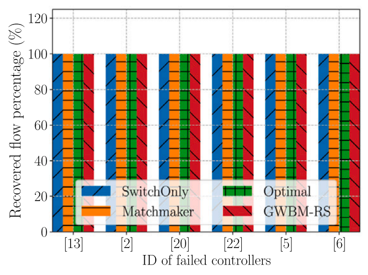
One controller failure: Fig. 8 shows the results of four algorithms when one of six controllers fails. There are 6 combinations of the one controller failure. Fig. 8(a) shows the percentage of programmable flows from offline switches. All four algorithms recover 100% flows from offline switches, and they remap the same number of offline switches to active controllers as shown in Fig. 8(c). In Fig. 8(b), all the four algorithms have adequate processing ability to recover the offline flows.

Two controller failures: Fig. 9 shows the results of four algorithms when two of six controllers fail. There are 15 combinations of the two controller failures.

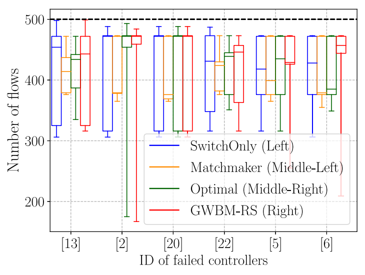
(1) Percentage of recovered flows: Fig. 9(a) shows the percentage of programmable flows from offline switches. SwitchOnly only recovers flows in the range of 40% to 86% because it cannot let all the offline switches to be remapped, so the offline flows traversing the switches, which remain unremapped, are still offline flows and cannot be recovered. Optimal performs best and recovers flows in the range of 87% to 95% because it smartly selects the proper path of flows to configure suitable switch-controller mapping relationship. Matchmaker

Table 2
Default relationship between controllers, switches, and the number of flows in the switches under ATT topology.

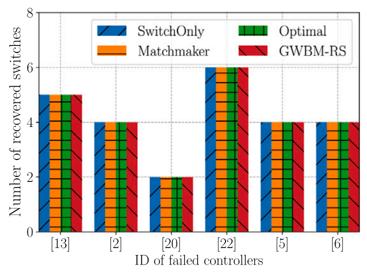
Controller ID	2				5				6				13					20		22					
Switch ID	2	3	9	16	4	5	8	14	0	1	6	7	10	11	12	13	15	19	20	17	18	21	22	23	24
Number of flows	143	71	107	55	49	143	53	61	81	49	89	97	63	59	71	213	67	49	63	125	49	81	111	49	57



(a) Percentage of recovered flows. The higher, the better.

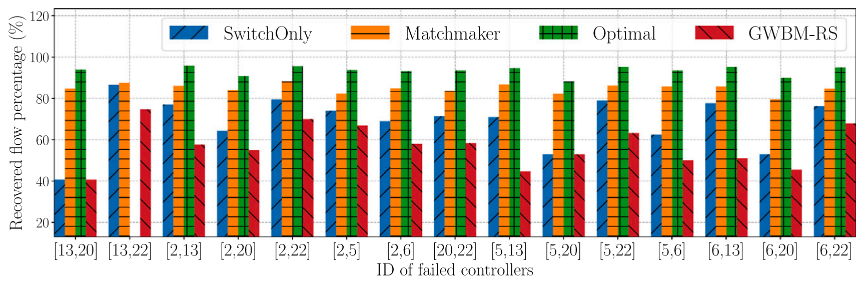


(b) Processing load of active controllers. The black dash line indicates the controller's processing ability.

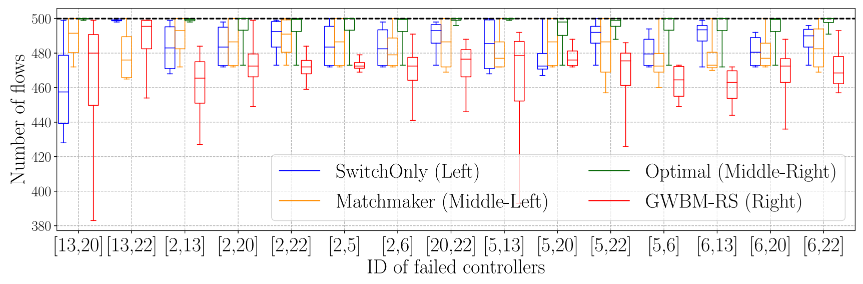


(c) The number of recovered offline switches. The higher, the better.

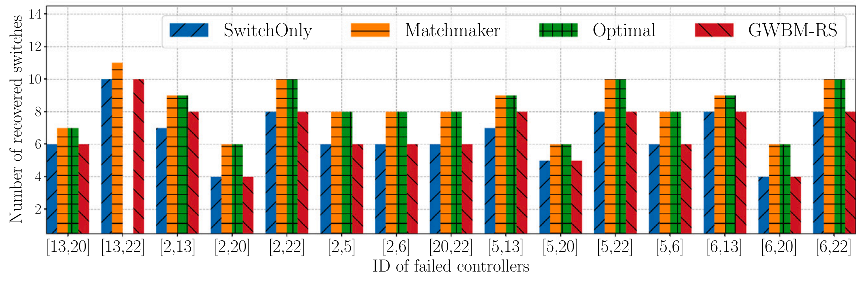
Fig. 8. Results of one controller failure under ATT topology.



(a) Percentage of recovered flows. The higher, the better.



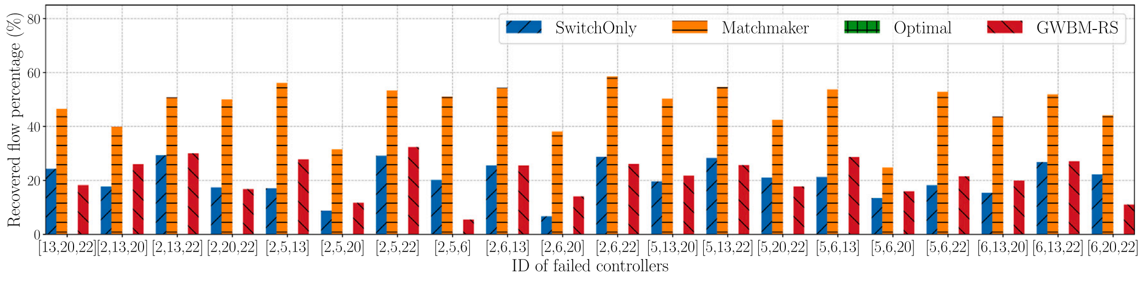
(b) Processing load of active controllers. The black dash line indicates the controller's processing ability.



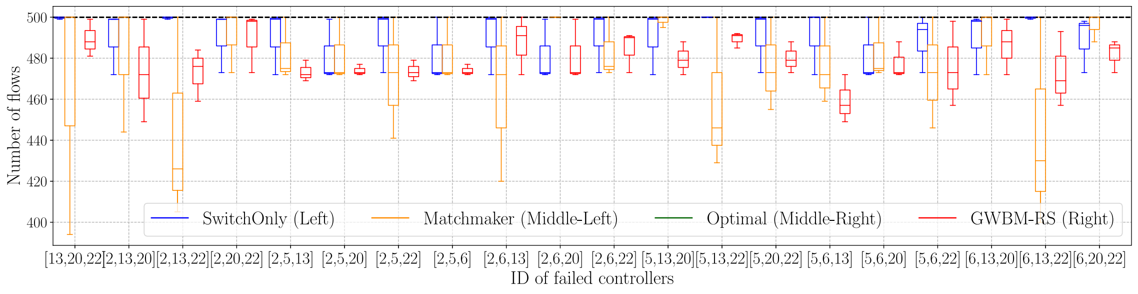
(c) The number of recovered offline switches. The higher, the better.

Fig. 9. Results of two controller failures under ATT topology. Optimal does not generate results in case [13, 22] due to its high complexity.

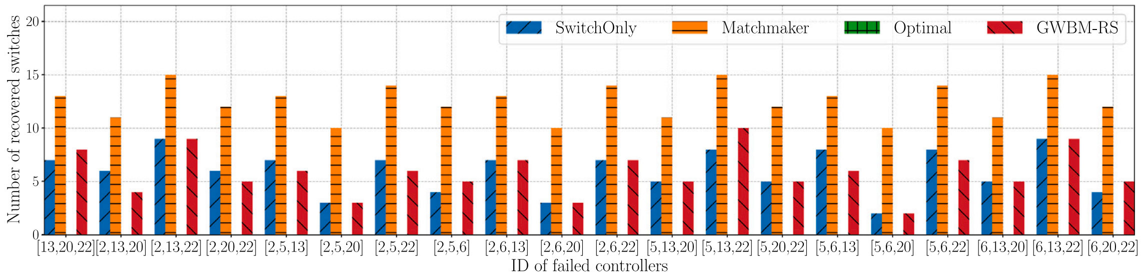
also performs well and recovers 79% to 87% offline flows. GWMB-RS performs worst since it is a static solution and only recovers offline flows in the range of 40% to 75%. For the special failure case (13, 20), Matchmaker's performance is 43% higher than SwitchOnly and GWBM-RS. Specifically, the control cost of offline switch s_{13} is 213, but the available control resource of active controllers C_2, C_5, C_6, and



(a) Percentage of recovered flows. The higher, the better.



(b) Processing load of active controllers. The black dash line indicates the controller's processing ability.



(c) The number of recovered offline switches. The higher, the better.

Fig. 10. Results of three controller failures under ATT topology. Optimal does not generate results due to its high complexity.

C_{22} are 124, 194, 184, and 28, respectively. Thus, switch s_{13}'s control cost cannot match the control resource of any controllers, and thus it cannot be recovered. On the contrary, Matchmaker can remap switch s_{13} to controller C_5 by dynamically altering the control cost of switch s_{13} based on the control resource of active controllers.

(2) Processing load of active controllers: Fig. 9(b) shows the processing load of active controllers. Optimal performs best in all the cases since it fully uses the available resources of active controllers with high efficiency. Matchmaker also performs well in utilization efficiency of active controllers because it uses the least control resource to realize good recovery performance of offline flows, as shown in Fig. 9(a). The available resources in SwitchOnly and GWBM-RS are under low utilization efficiency because they use more control resource to recover less offline flows compared with Matchmaker.

(3) The number of recovered switches: Fig. 9(c) shows the number of recovered offline switches. Matchmaker and Optimal recover all the offline switches because they smartly select the proper path of flows. SwitchOnly and GWBM-RS perform worst because the available control ability cannot guarantee to recover all the offline switches using fixed routing paths.

Three controller failures: Fig. 10 shows the results of four algorithms when three of six controllers fail. There are 20 combinations of the three controllers failures. Note that Optimal did not get the results due to the huge time cost.

(1) Percentage of recovered flows: Fig. 10(a) shows the percentage of programmable flows from offline switches. SwitchOnly only recovers flows in the range of 7% to 29% because the more switches become offline, the worse efficiency SwitchOnly performs. Matchmaker performs much better than SwitchOnly and GWBM-RS and recovers 26% to 58% offline flows. Matchmaker's performance is 45% higher than GWBM-RS. GWBM-RS performs worst and only recovers offline flows in the range of 5% to 32%. This result indicates that static solution neglects the different control cost of offline switches. On the contrary, Matchmaker enjoys the adaptive control cost of switches by smartly changing the paths of flows.

(2) Processing load of active controllers: Fig. 10(b) shows the processing load of active controllers. Matchmaker performs best since it uses the least controller resource to realize the best performance in recovering offline flows, as shown in Fig. 10(a). SwitchOnly and GWBM-RS perform worst since they utilize the control resource of active controllers under low efficiency.

(3) The number of recovered switches: Fig. 10(c) shows the number of recovered offline switches. Matchmaker recovers all the offline switches by smartly selecting the proper paths of flows. SwitchOnly and GWBM-RS perform worst because the available control ability cannot guarantee to recover all the offline switches using fixed routing paths.

6.3.2. Under Belnet topology

One controller failure: Fig. 11 shows the results of four algorithms when one of five controllers fails. There are 5 combinations of the one controller failure. Fig. 11(a) shows the percentage of programmable flows from offline switches. All four algorithms recover 100% flows from offline switches in majority of cases, and they remap the same number of offline switches to active controllers in these four cases as

Table 3
Default relationship between controllers, switches, and the number of flows in the switches under Belnet topology.

Controller ID	1				4					11			17				18				
Switch ID	0	1	3	6	4	5	8	9	12	2	10	11	7	13	17	20	14	15	16	18	19
Number of flows	65	89	53	113	117	91	63	49	61	65	113	89	87	59	209	55	43	59	91	243	59

shown in Fig. 11(c). For the special failure case 18, Matchmaker's performance is 77% higher than SwitchOnly and GWBM-RS. Specifically, the control cost of offline switch s18 is 243, but the available control resource of active controllers C_1, C_4, C_{11}, and C_{17} are 180, 119, 233, and 88, respectively. Thus, switch s_{18}'s control cost cannot match the control resource of any controllers, and thus it cannot be recovered. On the contrary, Matchmaker can remap switch s_{18} to controller C_{11} by dynamically altering the control cost of switch s_{18} based on the control resource of active controllers. In Fig. 11(b), SwitchOnly and GWBM-RS cannot efficiently utilize the control resource of active controllers, while Matchmaker and Optimal make full use of available control resource by adaptively altering the control cost of switches.

Two controller failures: Fig. 12 shows the results of four algorithms when two of five controllers fail. There are 10 combinations of the two controller failures.

(1) Percentage of recovered flows: Fig. 12(a) shows the percentage of programmable flows from offline switches. Optimal performs the best, it recovers flows in the range of 56% to 77%, because it smartly selects the proper path of flows configures suitable switch-controller mapping relationship. However, SwitchOnly performs the worst and only recovers flows in the range of 2% to 28% because it cannot let all the offline switches be remapped, so the offline flows traversing the switches which remain unremapped are still offline flows and cannot be recovered. Matchmaker also performs well, it recovers 48% to 65% offline flows, and increases the number of recovered flows up to 50%. GWBM-RS recovers the offline flows at the range of 12% to 34%.

(2) Processing load of active controllers: Fig. 12(b) shows the processing load of active controllers. SwitchOnly and GWBM-RS performs the worst since they utilize the control resource of active controllers under low efficiency. While Optimal performs the best because it fully utilize the control resource and outperforms other solutions in recovering offline flows. Matchmaker performs the best since it uses the least controller resource to realize the good performance in recovering offline flows, which is shown in Fig. 12(a).

(3) The number of recovered switches: Fig. 12(c) shows the number of recovered offline switches. SwitchOnly and GWBM-RS perform the worst because the available control ability cannot recover all the offline switches. While Matchmaker and Optimal recover all the offline switches because they smartly select the proper path of flows and discard some of the flows when necessary.

6.3.3. *Computation time*

To show the computation efficiency, we evaluate the computation time of Matchmaker and Optimal under two controller failure scenarios and two topologies, and Fig. 13 shows the result. In the figure, the computation time of Matchmaker is only less than 1 s under one controller failure scenario and less than 10 s under two controller failures scenario. The computation time of Optimal grows to almost 1000 s under two controller failures scenario of both topologies. Thus, considering recovery performance in Figs. 8, 9, 11, and 12, the result indicates that the proposed Matchmaker can achieve good recovery performance with low computation time.

Matchmaker is a heuristic algorithm performing well in recovering offline flows under multiple controller failures. The optimal solution of the FRSM problem is to get its optimal result with IP optimization solvers. However, as the network size increases and the failure cases become severer, the solution space could increase significantly, and finding a feasible solution may need a very long time or perhaps is impossible, as shown in Fig. 13. For example, Optimal cannot get the results due to the high complexity of proposed FRSM problem, which are shown in the case [13, 22] of Fig. 9 and all cases of Fig. 10. As a result, we achieve trade-off between the performance and time complexity by solving the problem with the proposed heuristic algorithm Matchmaker.

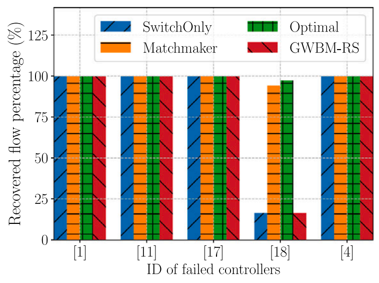
7. Related work

7.1. *Network resiliency*

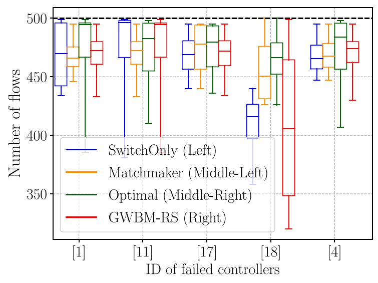
To maintain resiliency of the network, POCO [15], a framework for resilient Pareto-based Optimal COntroller-placement, is proposed to enable the operator of a network to make all Pareto-optimal placements. Vizarreta et al. [22] present two strategies to address the Reliable Controller Placement (RCP) problem, which protect the control plane against link and node failures. Mohammed et al. [23] present a new metric to measure node's importance to determine the locations of controllers against network failure. Simulated Annealing Strategy for Reliable Controller Placement (SASRCP) [16] is proposed to maintain network resiliency which can Reduce the propagation latency and limit the routing cost of the network under controller failures. Mohan et al. [24] introduce a novel primary-backup controller mapping approach in which switches are mapped to backup controllers to counteract controller failures that may happen simultaneously. Hu et al. [11] find that the unreasonable slave controller assignment could lead to the controller chain failure and possibly crash the entire network in the end. They then consider some significant factors for the design of fault-tolerant control plane and formulate it as Slave Controller Assignment (SCA) problem.

7.2. *Controller-switch mapping*

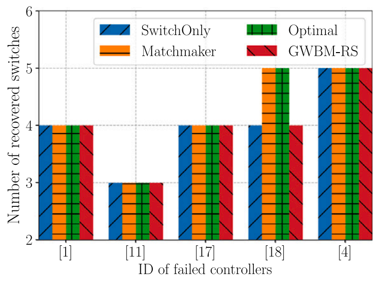
Tanha et al. [5] propose to choose the optimal placement of controllers, which takes both the switch-controller and inter-controller latency requirements and the capacity of the controllers into consideration. RetroFlow [17] leverages the features of hybrid routing in SDN switches to maintain the flow programmability and low communication overhead without overloading active controllers. He et al. [8] propose a master and slave controller assignment model, which can prevent multiple controller failures under the constraints of propagation latency. Acan et al. [25] formulate a reactive assignment model against network failures using integer linear programming based on load distribution of controllers under switch, link and controller failures. DCAP [26], a novel controller assignment approach, is proposed to minimize the average response time of the control plane in data center networks. They also reformulate DCAP as an online optimization to minimize the total cost [27]. Huang et al. [28] propose a novel scheme to jointly consider both static and dynamic switch-controller association and devolution. ProgrammabilityGuardian [29] improves the path programmability of offline flows and maintains low communication overhead by using a middle layer to establish the fine-grained flow-controller mappings. Controller load distribution, reassignment cost, and probability of failure are all considered in PREFCP [30], which is a proactive switch assignment approach using a genetic algorithm based heuristic.



(a) Percentage of recovered flows. The higher, the better.

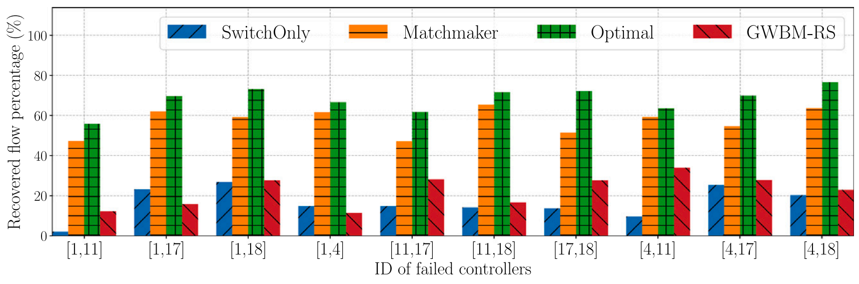


(b) Processing load of active controllers. The black dash line indicates the controller's processing ability.

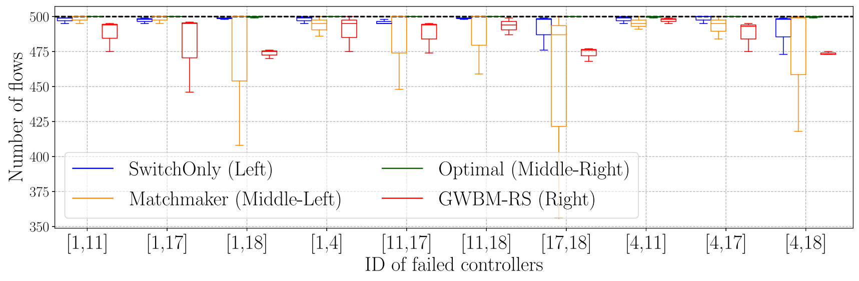


(c) The number of recovered offline switches. The higher, the better.

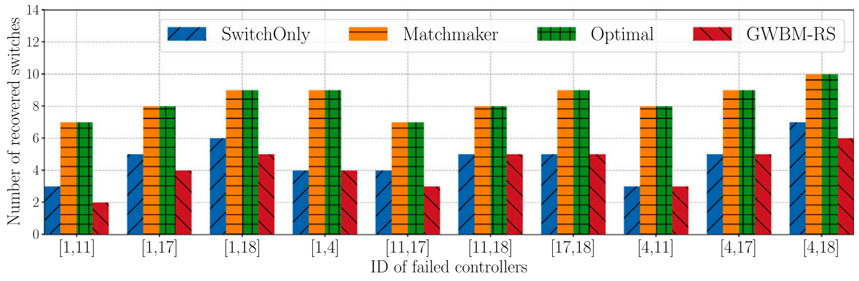
Fig. 11. Results of one controller failure under Belnet topology.



(a) Percentage of recovered flows. The higher, the better.



(b) Processing load of active controllers. The black dash line indicates the controller's processing ability.



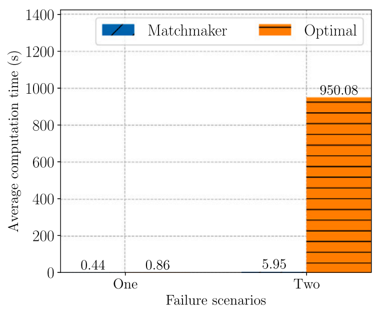
(c) The number of recovered offline switches. The higher, the better.

Fig. 12. Results of two controller failures under Belnet topology.

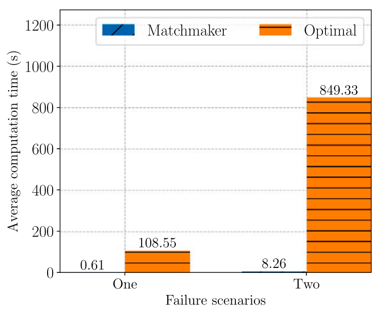
8. Conclusion and future directions

In this paper, we design Matchmaker to improve the programmability of offline flows due to controller failure. Matchmaker is an adaptable solution which can adaptively adjust the control cost of offline switches based on the limited control resource of active controllers by changing the paths of flows. Therefore, with the help of Matchmaker, more offline switches can be remapped to active controllers, and the number of recovered flows increases.

In future, we will consider impact of other failures in our work. For example, the control path between a controller and a switch may fail due to the failures of links and nodes, which leads to the failing of control for switches. To incorporate this failure into our model, we can consider failure probability for each connection between a controller and a switch, and other network failure situations involved in the control path. We can further update our Matchmaker for more failure scenarios.



(a) Under ATT topology.



(b) Under Belnet topology.

Fig. 13. Percentage of computation time of Matchmaker to Optimal. The lower, the better. In (a), Optimal needs 0.86 s on average.

CRediT authorship contribution statement

Songshi Dou: Writing - original draft, Methodology, Software. **Guochun Miao:** Formal analysis, Writing - review & editing. **Zehua Guo:** Conceptualization, Formal analysis, Writing - review & editing, Supervision, Funding acquisition. **Chao Yao:** Writing - review & editing, Funding acquisition. **Weiran Wu:** Visualization. **Yuanqing Xia:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

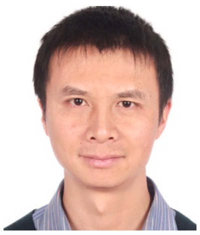
Acknowledgments

This paper was supported by the National Natural Science Foundation of China under Grants 62002019 and 61802315, and the Beijing Institute of Technology Research Fund Program for Young Scholars. The authors thank Weikun Chen for his help during the preparation of this paper.

References

[1] S. Jain, A. Kumar, S. Mandal, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, B4: Experience with a globally-deployed software defined WAN, in: ACM SIGCOMM'13.
[2] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven WAN, in: ACM SIGCOMM'13.
[3] T. Hu, Z. Guo, P. Yi, T. Baker, J. Lan, Multi-controller based software-defined networking: A survey, IEEE Access 6 (2018) 15980–15996.
[4] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, H.J. Chao, Improving the performance of load balancing in software-defined networks through load variance-based synchronization, Comput. Netw. 68 (2014) 95–109.
[5] M. Tanha, D. Sajjadi, J. Pan, Enduring node failures through resilient controller placement for software defined networks, in: IEEE GLOBECOM'16.
[6] M. Tanha, D. Sajjadi, R. Ruby, J. Pan, Capacity-aware and delay-guaranteed resilient controller placement for software-defined WANs, IEEE Trans. Netw. Serv. Manag. (2018) 1.
[7] N. Perrot, T. Reynaud, Optimal placement of controllers in a resilient SDN architecture, in: IEEE DRCN'16.
[8] F. He, T. Sato, E. Oki, Master and slave controller assignment model against multiple failures in software defined network, in: IEEE ICC'19.
[9] N. van Adrichem, C. Doerr, F.A. Kuipers, OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks, in: IEEE NOMS'14.
[10] A. Tootoonchian, M. Ghobadi, Y. Ganjali, OpenTM: Traffic matrix estimator for OpenFlow networks, in: Springer PAM'10.
[11] T. Hu, Z. Guo, J. Zhang, J. Lan, Adaptive slave controller assignment for fault-tolerant control plane in software-defined networking, in: IEEE ICC'18.
[12] T. Hu, P. Yi, Z. Guo, J. Lan, Y. Hu, Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks, Future Gener. Comput. Syst. 95 (JUN.) (2019) 681–693.
[13] G. Yao, J. Bi, L. Guo, On the cascading failures of multi-controllers in Software Defined Networks, in: IEEE ICNP'14.
[14] B.P.R. Killi, S.V. Rao, Capacitated next controller placement in software defined networks, IEEE Trans. Netw. Serv. Manag. 14 (3) (2017) 514–527.
[15] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, P. Tran-Gia, POCO-framework for Pareto-optimal resilient controller placement in SDN-based core networks, in: IEEE ITC'13.
[16] S. Mohanty, K. Kanodia, B. Sahoo, K. Kurroliya, A Simulated Annealing Strategy for Reliable Controller Placement in Software Defined Networks, in: IEEE SPIN'20.
[17] Z. Guo, W. Feng, S. Liu, W. Jiang, Y. Xu, Z. Zhang, RetroFlow: Maintaining control resiliency and flow programmability for software-defined WANs, in: IEEE/ACM IWQoS'19.
[18] OpenFlow Switch Specification 1.3, https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf.
[19] D.J. Watts, S.H. Strogatz, Collective dynamics of 'small-world' networks, Nature (1998).
[20] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo, IEEE J. Sel. Areas Commun. 29 (9) (2011) 1765–1775.
[21] Gurobi Optimization, http://www.gurobi.com.
[22] P. Vizarreta, C.M. Machuca, W. Kellerer, Controller placement strategies for a resilient SDN control plane, in: IEEE RNDM'16.
[23] M.J.F. Alenazi, E.K. «etinkaya, Resilient placement of SDN controllers exploiting disjoint paths, Trans. Emerg. Telecommun. Technol. (2) (2019).
[24] P. Murali Mohan, T. Truong-Huu, M. Gurusamy, Byzantine-resilient controller mapping and remapping in software defined networks, IEEE Trans. Netw. Sci. Eng. (2020) 1.
[25] F. Açan, G. Gür, F. Alagöz, Reactive controller assignment for failure resilience in software defined networks, in: IEEE APNOMS'19.
[26] T. Wang, F. Liu, J. Guo, H. Xu, Dynamic SDN controller assignment in data center networks: Stable matching with transfers, in: IEEE INFOCOM'16.
[27] T. Wang, F. Liu, H. Xu, An efficient online algorithm for dynamic SDN controller assignment in data center networks, IEEE/ACM Trans. Netw. (2017) 1–14.
[28] H. Xi, S. Bian, Z. Shao, X. Hong, Dynamic switch-controller association and control devolution for SDN systems, in: IEEE ICC'17.
[29] Z. Guo, S. Dou, W. Jiang, Improving the Path Programmability for Software-Defined WANs under Multiple Controller Failures, in: IEEE/ACM IWQoS'20.
[30] S. Güner, G. Gür, F. Alagöz, Proactive controller assignment schemes in SDN for fast recovery, in: IEEE ICOIN'20.

Songshi Dou received the B.S. degree from the School of Control and Computer Engineering, North China Electric Power University, Beijing, China, in 2019. He is currently pursuing the M.S. degree in the School of Automation, Beijing Institute of Technology, Beijing, China. His research interests cover software-defined networking, network function virtualization and data center network.



Guochun Miao received the B.S. degree and the M.S. degree from University of Electronic Science and Technology of China, Chengdu, China. He was a Research Fellow at the Department of Electronic Science and Engineering. He worked at China Shipbuilding Industry Systems Engineering Research Institute, Beijing, China. He is a senior engineer at Haifeng General Aviation Technology Co., Ltd. His research interests include software-defined radio, big data analysis, electronic science and engineering, and systems engineering.

Zehua Guo received B.S. degree from Northwestern Polytechnical University, Xi'an, China, M.S. degree from Xidian University, Xi'an, China, and Ph.D. degree from Northwestern Polytechnical University. He was a Research Fellow at the Department of Electrical and Computer Engineering, New York University Tandon School of Engineering, New York, NY, USA, and a Research Associate at the Department of Computer Science and Engineering, University of Minnesota Twin Cities, Minneapolis, MN, USA. His research interests include programmable networks (e.g., software-defined networking, network function virtualization), machine learning, and network security. Dr. Guo is an Associate Editor for IEEE Systems Journal and the EURASIP Journal on Wireless Communications and Networking (Springer), and an Editor for the KSII Transactions on Internet and Information Systems. He is serving as the TPC of several journals and conferences (e.g., Elsevier Computer Communications, AAAI, IWQoS, ICC, ICCCN, ICA3PP). He is a Senior Member of IEEE, China Institute of Communications, Chinese Institute of Electronics, and a Member of ACM and China Computer Federation.

Chao Yao received his B.Sc. in telecommunications engineering in 2007, and the Ph.D. degree in communication and information systems in 2014, both from Xidian University, Xi'an, China. He was a visiting student in Center for Pattern Recognition and Machine Intelligence (CENPARMI), Montreal, Canada, during 2010–2011. His research interests include feature extraction, handwritten character recognition, machine learning, and pattern recognition.

Weiran Wu was born in Hubei, China, in 1999. At present he is a junior at the School of Automation, Beijing Institute of Technology, Beijing, China. His research interests cover computer network and the application of control theory in this field.

Yuanqing Xia was born in Anhui, China, in 1971. He received the M.E. degree from the School of Mathematical Sciences, Anhui University, Hefei, China, in 1998, and the Ph.D. degree from the School of Automation Science and Electrical Engineering, Beihang University, Beijing, China, in 2001. From 2002 to 2003, he was a Postdoctoral Research Associate with the Institute of Systems Science, Academy of Mathematics and System Sciences, Chinese Academy of Sciences, Beijing. From 2003 to 2004, he was a Research Fellow with the National University of Singapore, Singapore, where he researched on variable structure control. From 2004 to 2006, he was a Research Fellow with the University of Glamorgan, Pontypridd, U.K. From 2007 to 2008, he was a Guest Professor with Innsbruck Medical University, Innsbruck, Austria. Since 2004, he has been with the Department of Automatic Control, Beijing Institute of Technology, Beijing, first as an Associate Professor, then, since 2008, as a Professor. His current research interests include the fields of networked control systems, robust control and signal processing, and active disturbance rejection control.