Toward Flexible and Predictable Path Programmability Recovery Under Multiple Controller Failures in Software-Defined WANs

Zehua Guo, *Senior Member, IEEE, Member, ACM*, Songshi Dou, *Graduate Student Member, IEEE, ACM*, Wenfei Wu, *Member, IEEE*, and Yuanqing Xia, *Senior Member, IEEE*

***Abstract*— Software-Defined Networking (SDN) promises good network performance in Wide Area Networks (WANs) with the logically centralized control using physically distributed controllers. In Software-Defined WANs (SD-WANs), maintaining path programmability, which enables flexible path change on flows, is crucial for maintaining network performance under traffic variation. However, when controllers fail, existing solutions are essentially coarse-grained switch-controller mapping solutions and only recover the path programmability of a limited number of offline flows, which traverse offline switches controlled by failed controllers. In this paper, we propose FlexibleProgrammabilityMedic (FlexPM) to provide predictable path programmability recovery under multiple controller failures in SD-WANs. The key idea of FlexPM is to approximately realize flow-controller mappings using hybrid SDN/legacy routing supported by high-end commercial SDN switches. Using the hybrid routing, we can recover programmability by selecting a routing mode for each offline flow at each offline switch in a fine-grained way to fit the given control resource from active controllers and release a few control resource of active controllers by reasonably configuring some normal flows under legacy routing mode. Thus, FlexPM can promise ample control resource to improve the recovery efficiency and further effectively map offline switches to active controllers. Simulation results show that FlexPM outperforms existing switch-level solutions by maintaining balanced programmability and increasing the total programmability of recovered offline flows up to 660% under AT&T topology and 590% under Belnet topology.**

***Index Terms*— software-defined networking, wide area networks, controller failure, path programmability.**

I. Introduction

WIDE Area Network (WAN) is a crucial infrastructure in real world. It connects and transfers traffic among different types of networks (*e.g.*, data centers, cellular networks, metropolitan area networks). To improve performance, emerging Software-Defined Networking (SDN) has been deployed in WANs, known as the SD-WANs. Microsoft SWAN [2] and Google B4 [3] show that flexible flow control enabled by SDN can significantly improve the utilization of WANs. Basically, an SD-WAN has a large scale and is usually composed of multiple domains, each of which consists of one SDN controller to control SDN switches placed in its domain. The flexible flow control of SDN comes from *path programmability*. Once a flow traverses an SDN switch, it becomes a *programmable flow* and is able to change its forwarding path.

Path programmability is empowered by the SDN controller through deploying flow entries to adjust flows' forwarding paths. A controller is a network control software installed in a physical server or a virtual machine. Because of unpredictable issues (*e.g.*, hardware/software bugs, power failure), the controller may fail, and its controlled switches will be unmanageable and become *offline*. In this case, flows, which traverse offline switches, lose their path programmability to change their paths and become *offline flows*. As a result, the network suffers from path programmability degradation and potentially experiences performance fluctuation under traffic variation. Therefore, recovering path programmability for offline flows under controller failure is the key to guaranteeing network performance. Maintaining path 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. Several controllers may fail simultaneously or fail successively. Some existing works [4], [5], [6], [7], [8] also study the problem of multiple controller failures. However, existing solutions suffer from poor recovery performance since the available control resource of active controllers is limited, and the fixed control cost of switches does not always match controllers' control resource properly.

Programmability recovery is realized by remapping the control of offline switches to active controllers to let offline flows in offline switches become programmable again. The main challenge of this mapping problem is how to use the limited control resource from active controllers to handle the control cost for enabling the programmability of offline flows under different controller failures. Without

Manuscript received 14 November 2021; revised 24 April 2022, 14 July 2022, and 15 November 2022; accepted 29 November 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. C.-J. Lin. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB1714800, in part by the National Natural Science Foundation of China under Grant 62002019, in part by the Zhejiang Lab Open Research Project under Grant K2022QA0AB02, in part by the SongShan Laboratory Fund under Grant YYJC022022009, and in part by the Beijing Institute of Technology Research Fund Program for Young Scholars. This paper was presented in part at IEEE ICDCS 2021 [1] [DOI: 10.1109/ICDCS51616.2021.00051]. *(Corresponding author: Songshi Dou.)*

Zehua Guo, Songshi Dou, and Yuanqing Xia are with the Beijing Institute of Technology, Beijing 100081, China (e-mail: songshidou@hotmail.com).

Wenfei Wu is with the Peking University, Beijing 100871, China.

Digital Object Identifier 10.1109/TNET.2022.3227423

appropriate remapping, active controllers could be overloaded to recover offline flows, and network performance may degrade. In the worse case, network may face significant performance degradation due to the cascading controller failure [9].

Existing solutions can be categorized into two classes: switch-level solutions and flow-level solutions. Switch-level solutions consider all flows in a switch as an unit and remap offline switches to active controllers based on the switches' and controllers' current status information [4], [6], [7], [8], [10]. The coarse-grained switch-controller mapping only recovers a small number of offline flows from offline switches because the fixed control cost of switches does not always match controllers' control resource properly. A recent flow-level solution proposes to remap offline flows to active controllers in a fine-grained way by introducing a middle layer between controllers and switches [11]. This solution performs well in terms of programmability recovery, but the middle layer design brings new issues (*e.g.*, increasing the processing delay [12], bringing new unreliability).

In this paper, we propose FlexibleProgrammabilityMedic (FlexPM), a novel switch-controller mapping solution to improve the programmability recovery of offline flows. The key of FlexPM is to take advantage of the hybrid SDN/legacy routing mode supported by high-end commercial SDN switches to approximately realize fine-grained flow-controller mappings. The hybrid routing employs a high-priority flow table using OpenFlow and a low-priority legacy routing table based on OSPF. With the proper configuration, we can jointly use the two tables in a sequential fashion to route flows in different modes and thus recover the path programmability in a fine-grained per-flow fashion based on the given control resource of active controllers. However, the limited control resource of active controllers becomes the key bottleneck factor for recovering offline flows. To increase the control resource for recovering flows, we choose some normal flows on online switches working under the legacy mode to release a few control resource for controlling normal flows. We carefully select the proper normal flows to work under the legacy mode without interrupting their normal operations. Consequently, as offline switches efficiently mapped to the active controllers with more available control resource, the programmability of offline flows is effectively recovered.

The contributions of this paper are summarized as follows:

- We propose to recover path programmability of offline flows by approximately realizing flow-controller mappings using hybrid SDN/legacy routing supported by high-end commercial SDN switches.
- We formulate the Joint Flow Mode Selection and Switch Mapping (JFMSSM) problem, which uses hybrid routing and aims to recover offline flows with balanced programmability by deciding the routing mode of flows at offline switches and the mappings between offline switches and active controllers, and further improves the control resource of active controllers by configuring some normal flows under the legacy routing mode at online switches to release the control resource for controlling these normal flows.
- The proposed JFMSSM problem is a Mixed-Integer Programming with high computation complexity. To efficiently solve the problem, we reformulate the problem with linear techniques and solve the problem with the proposed efficient heuristic algorithm named FlexPM.
- We evaluate the performance of FlexPM under different controller failure scenarios. The results show that FlexPM outperforms existing switch-level solutions by maintaining balanced programmability and increasing the total programmability of offline flows up to 660% under AT&T topology and 590% under Belnet topology.

The rest of the paper is organized as follows. In Section II, we introduce the background and motivation of this paper. Section III presents our design considerations. Section IV mathematically formulates our programmability recovery problem as the JFMSSM problem. Section V proposes FlexPM to efficiently solve the problem. We evaluate and analyze the performance of FlexPM in Section VI. Section VII introduces related work, and Section VIII concludes this paper.

II. Background and Motivation

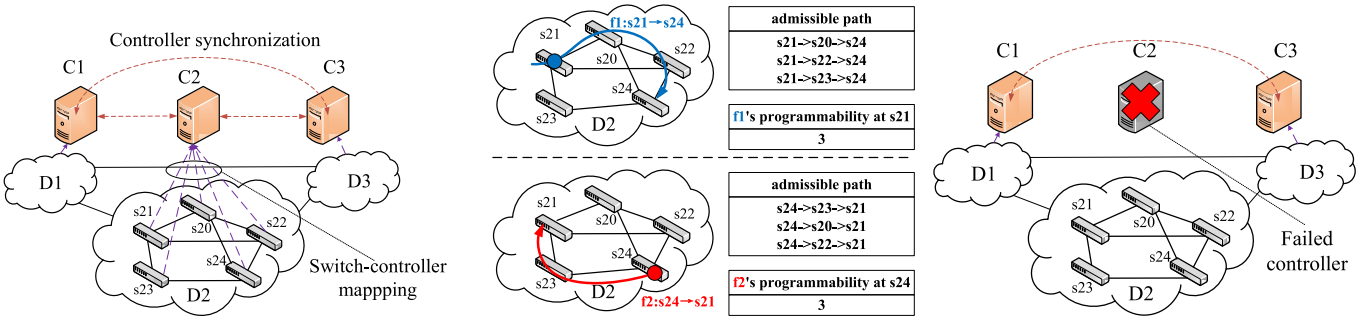
A. Network Control and Path Programmability in SD-WANs

We use Fig. 1 to explain the SD-WAN. In Fig. 1(a), the SD-WAN consists of three domains, and the SDN control plane includes three controllers, each of which controls one domain and synchronizes with others to maintain a logically centralized network view. For simplicity, we only concentrate on domain D_2, which has five SDN switches $s_{20} - s_{24}$ and is controlled by controller C_2.

SDN introduces path programmability to improve network performance by dynamically rerouting flows under traffic variation. If a flow traverses an SDN switch, the switch can only change the next hop on the path of the flow. The admissible path of a flow in a switch is the feasible path that a flow can use from the switch's next hop to the flow's destination. Thus, the path programmability of one flow at a switch is denoted as the number of admissible paths from the switch's next hops to the flow's destination. Fig. 1(b) is an example to illustrate the path programmability of two flows. In this example, we only consider the shortest paths for each flow when calculating the number of admissible paths. For flow f^1:$s_{21} \rightarrow s_{24}$, it has three admissible paths (*i.e.*, $s_{21} \rightarrow s_{20} \rightarrow s_{24}$, $s_{21} \rightarrow s_{22} \rightarrow s_{24}$, and $s_{21} \rightarrow s_{23} \rightarrow s_{24}$) at s_{21}, and thus its programmability is three at s_{21}. Similarly, the programmability of flow f^2: $s_{24} \rightarrow s_{21}$ is three at s_{24} since it has three admissible paths (*i.e.*, $s_{24} \rightarrow s_{23} \rightarrow s_{21}$, $s_{24} \rightarrow s_{22} \rightarrow s_{21}$, and $s_{24} \rightarrow s_{20} \rightarrow s_{21}$) at s_{24}.

However, SDN controllers may fail unpredictably. Under controller failures, SDN switches controlled by failed controllers become *offline*, and flows, which traverse offline switches, become offline and lose their programmability to change their paths to cope with network variation. As a result, it is difficult to maintain resilient network control. We use Fig. 1(c) to illustrate this issue. In Fig. 1(c), controller C_2 fails, and its controlled five switches in D_2 become offline. Thus, f^1 and f^2 become offline flows.

If the controller loses the control over some switches, these switches will become offline. These offline switches



(a) An SD-WAN. (b) Path programmability of two flows. (c) Controller failure in an SD-WAN.

Fig. 1. Example of an SD-WAN and the path programmability. A circle denotes that a flow is programmable at this switch.

can forward packets using two methods: (1) following the flow entries installed by the controller before it fails and (2) checking the default traditional routing table and using the destination-based entries. Both the two methods can be viewed as static routing, and flows are fixedly forwarded on their paths in spite of network variation. The controller cannot flexibly change the flow's path at the offline switch, and the flow has only one default routing path to the destination at the offline switch. Thus, all flows that traverse offline switches will lose the path programmability because flexibly rerouting flows at these switches is impossible without the controller. In this case, offline switches do not contribute extra routing paths to the path programmability, thus they do not contribute to the path programmability of flows.

In SD-WANs, the key of maintaining resilient network control is to enable the path programmability. To recover the path programmability, we must hand over the control of switches $s_{20} - s_{24}$ to active controllers C_1 and C_3. A feasible solution to recover path programmability of offline flows is to remap offline switches to active controllers. In this way, all the offline flows that traverse remapped switches are re-controlled by active controllers and become programmable again.

Note that we do not have to let all switches be programmable to ensure that each flow has the path programmability. If a flow traverses both SDN switches and offline switches, the programmability of this flow is calculated based on the programmability at all SDN switches. Offline switches are not considered while calculating the path programmability since we cannot change the path of a flow at these offline switches, and the flow does not possess any path programmability at these offline switches. As a result, we do not consider flows' programmability at offline switches in our paper, and the programmability value is defined as zero at offline switches. We can selectively choose proper switches becoming programmable to improve path programmability of the whole network. However, with more SDN switches to be programmable, the path programmability of the whole network will increase.

B. Existing Solutions and Their Limitations

Existing solutions for recovering path programmability can be categorized into the following two classes:

1) Switch-Level Solutions: These solutions employ the default path programmability recovery solution originated from OpenFlow [13] to remap offline switches to active controllers. By remapping one offline switch to an active controller, all flows that traverse this remapped switch are controlled by this controller and become programmable again [4], [6], [14]. Switch-level solutions do not work well under multiple controller failures due to their coarse-grained mappings. To mitigate the impact of coarse-grained mapping, a recent work proposes a hybrid switch mode solution [10], which reduces the control cost of offline switches by configuring them with different routing modes (*i.e.*, SDN and legacy) and only remapping switches with the SDN mode to controllers. This solution improves recovery efficiency but still suffers from coarse-grained mapping under serious controller failures.

2) Flow-Level Solutions: ProgrammabilityGuardian (PG) [11] is a flow-level solution. Instead of remapping all flows in an offline switch to an active controller, PG introduces a middle layer between controllers and switches using existing SDN slicing techniques (*i.e.*, FlowVisor [12]) and uses the layer to remap each offline flow to one active controller in a fine-grained manner. This flow-level solution performs much better than switch-level solutions but relies on an extra middle layer, which is usually realized by physical servers or virtual machines. The layer needs extra time to process incoming requests [12] and also suffers from unpredictable issues (*e.g.*, hardware/software bugs, power failure). Thus, this layer not only increases the processing delay but also brings new unreliability to the system.

III. Design Considerations of FlexPM

In this section, we introduce the opportunity and some key design considerations of FlexPM.

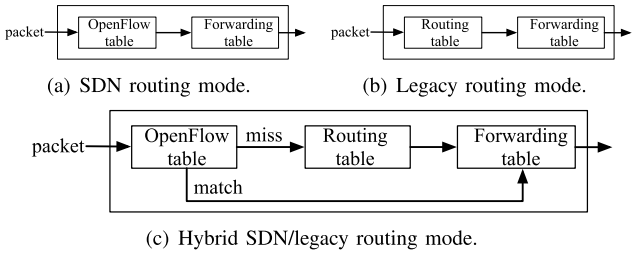
A. Opportunity

Based on the above analysis, we can see that the ideal recovery solution should be able to realize good recovery performance similar to flow-level solutions without extra devices. We can adapt our solution in two scenarios: for hybrid SDN/legacy devices and for OpenFlow-only devices.

1) Hybrid SDN/Legacy Devices: For the hybrid SDN/legacy devices, high-end commercial SDN switches provide a new opportunity for designing this solution. Brocade MLX-8 PE [15] supports different routing modes: SDN model, legacy

TABLE I
COMPARISON OF FLEXPM AND EXISTING SOLUTIONS

Solution	Mapping granularity	Extra device	Routing mode	Routing mode granularity	Recovery performance
switch-level solution	switch-controller	no	SDN, hybrid	per-switch	mediocre
flow-level solution	flow-controller	yes	SDN	per-flow	optimal
FlexPM	switch-controller	no	hybrid	per-flow	near optimal



(a) SDN routing mode. (b) Legacy routing mode.

(c) Hybrid SDN/legacy routing mode.

Fig. 2. Routing modes in high-end commercial SDN switches.

mode, and hybrid SDN/legacy routing mode, specifically OpenFlow/OSPF routing mode. Fig. 2 shows the data packet process of the three modes in the SDN switch. In Figs. 2(a) and 2(b), either OpenFlow or OSPF routing mode is used. In Fig. 2(c), OpenFlow and OSPF run at the same time, and the priority of the flow table used for OpenFlow is higher than that of the legacy routing table used by OSPF. The flow table can be installed with a default low-priority entry, which sends each unmatched packet to the legacy routing table. When the switch works in this hybrid mode, it will check the flow table for each incoming packet first. If a matched entry is found, the packet is processed according to the operation in the entry. Otherwise, the legacy routing table is then checked to route the packet using destination-based entries. The OpenFlow table and OSPF routing table share the same physical memory in the switch. Based on the configuration, the physical memory can be dynamically logically divided into OpenFlow table and legacy routing table. The hybrid routing mode is tested in production networks (*e.g.*, CHN-IX [16]). Some high-end commercial SDN switches may also support hybrid OpenFlow/OSPF mode, and existing works also build up the co-existing control-planes framework with detailed coexisting control-planes [17] and use hybrid routing mode to realize the improvement of network performance [18].

Using this hybrid mode, we can change the control cost of offline switches by selectively deciding the routing mode for each offline flow at each offline switch, and thus realize fine-grained path programmability recovery for each offline flow and improve the recovery efficiency. We use Table I to present the difference between existing solutions and our solution.

2) OpenFlow-Only Devices: If the hybrid SDN/legacy routing mode is not supported by the SDN switch, our proposed FlexPM can also be implemented. For these OpenFlow-only devices, we can first pre-configure some dedicated flow tables to realize the shortest path, and then use these flow tables to realize prefix-based forwarding. Note that these pre-configured flow tables are calculated at the beginning of the network initialization by OSPF. These shortest path-based entries are assigned with low priority, while other entries have high priority. Flows are forwarded based on the low-priority shortest path-based entries unless high-priority entries are deployed. Thus, we can realize hybrid routing in OpenFlow-only devices.

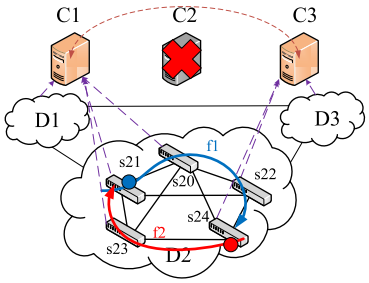
B. Design Considerations

To efficiently recover the path programmability, our design should consider the following three objectives:

- **Maximizing the number of recovered offline flows**: recovering path programmability of offline flows affects network performance and is the first priority for our problem. However, if some offline flows are not recovered, and these flows' traffic load vary significantly, these flows cannot be adaptively rerouted, and network performance could significantly fluctuate. Thus, we should maximize the number of recovered offline flows.
- **Balancing path programmability**: this consideration is to enable balanced path programmability of recovered flows. Generally speaking, a flow with a long path usually has higher programmability than a flow with a short path because the longer path increases the control probability of the flow. The path length implicitly distinguishes the priority of the flow and leads to an imbalanced path programmability. However, the size of flows may change over time, and unbalanced path programmability becomes another uncertainty for network performance. Thus, we should treat each offline flow equally by recovering each offline flow with similar programmability.
- **Fully utilizing controllers' control resource**: this consideration compensates for the above two considerations. Due to various conditions (*e.g.*, topology difference), in some cases, an offline flow can only be recovered with low programmability. Following the first and second considerations, other offline flows are also recovered with the same low programmability even if there is larger room to improve the programmability of these flows by establishing more mappings. Hence, we should fully utilize controllers' control resource to improve the overall programmability of the network.

C. An Example

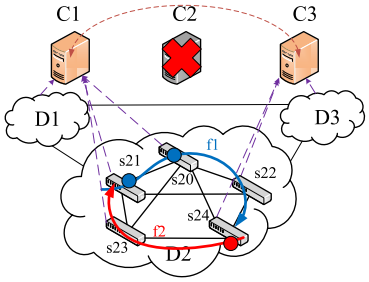
We use Fig. 3 as an example to illustrate the three considerations for designing FlexPM under the same controller failure in Fig. 1(c). Fig. 3(a) shows the result of a hybrid flow routing-based solution. In this figure, switches s_{20}, s_{21}, and s_{23} are mapped to active controller C_1, while other two switches are remapped to controller C_3. Flow f^1 is configured with SDN mode at switch s_{21}, flow f^2 is recovered at switch s_{24} with SDN mode. Flow f^1's programmability at switch s_{21} can be calculated as the number of admissible paths from s_{21}'s next hops to flow f^1's destination with the shortest distance,



flow	programmability
f1	3
f2	3

controller	control resource
C1	1
C3	0

(a) The result of only recovering offline flows with balanced programmability.



flow	programmability
f1	5
f2	3

controller	control resource
C1	0
C3	0

(b) The result of FlexPM, which recovers offline flows with balanced programmability and maximizes the total programmability.

Fig. 3. An example that illustrates design considerations. A colorful circle denotes that the flow with the same color works under SDN mode and is programmable at the switch.

and these paths are $s_{21} \rightarrow s_{20} \rightarrow s_{24}$, $s_{21} \rightarrow s_{22} \rightarrow s_{24}$, and $s_{21} \rightarrow s_{23} \rightarrow s_{24}$. For flow f^2, similar to flow f^1, all the admissible paths from s_{24}'s next hops to flow f^2's destination are $s_{24} \rightarrow s_{22} \rightarrow s_{21}$, $s_{24} \rightarrow s_{20} \rightarrow s_{21}$, and $s_{24} \rightarrow s_{23} \rightarrow s_{21}$. Thus, this solution can maintain balanced programmability of recovered flows because all the flows are recovered with the programmability of 3. However, in this figure, controller C_1 is not fully utilized.

Fig. 3(b) shows the result of FlexPM, which realizes three objectives. Compared with Fig. 3(a), flow f^1 is configured with SDN mode at switch s_{20} as shown in Fig. 3(b). Since flow f^1's forwarding paths can be adjusted at switches s_{21} and s_{20}, the increased paths from the recovery of switch s_{21} can be viewed as $s_{21} \rightarrow \boldsymbol{s_{20}} \rightarrow \boldsymbol{s_{22}} \rightarrow \boldsymbol{s_{24}}$ and $s_{21} \rightarrow \boldsymbol{s_{20}} \rightarrow \boldsymbol{s_{23}} \rightarrow \boldsymbol{s_{24}}$. Thus, the control resource of controller C_1 is fully utilized since FlexPM maximizes the total path programmability, and flow f^1 has the ability to flexibly change its path at switch s_{20} to further improve the network performance under traffic variation.

IV. Problem Formulation

In this section, we formulate the JFMSSM problem to decide the routing mode of offline flows at offline switches and remap offline switches to active controllers.

A. System Description

Typically, an SD-WAN consists of H distributed controllers, and each controller controls a domain of switches. The set of offline switches controlled by the failed controllers $\{C_{M+1}, \ldots, C_H\}$ are $\mathcal{S} = \{s_1, \ldots, s_i, \ldots, s_N\}$. The set of active controllers is $\mathcal{C} = \{C_1, \ldots, C_j, \ldots, C_M\}$, and the set of online switches controlled by the active controllers $\{C_1, \ldots, C_j, \ldots, C_M\}$ are $\mathcal{S}' = \{s_{N+1}, \ldots, s_t, \ldots, s_T\}$. D_{ij} denotes the propagation delay between offline switch s_i ($i \in [1, N]$) and controller C_j ($j \in [1, M]$). The set of flows traversing the set of offline switches $\mathcal{S}$ is $\mathcal{F} = \{f^1, f^2, \ldots, f^l, \ldots, f^L\}$. We use β_i^l and ζ_i^l to denote the relationship between switches and flows. If the forwarding path of flow f^l ($l \in [1, L]$) traverses offline switch s_i, and s_i has at least two paths to f^l's destination, we have $\beta_i^l = 1$; otherwise $\beta_i^l = 0$. Similarly, if the forwarding path of flow f^l ($l \in [1, L]$) traverses online switch s_t ($t \in [N+1, T]$), and s_t has at least two paths to f^l's destination, we have $\zeta_i^l = 1$; otherwise $\zeta_i^l = 0$. We use $x_{ij} = 1$ to denote that offline switch s_i ($i \in [1, N]$) is mapped to controller C_j; otherwise $x_{ij} = 0$. If flow f^l's forwarding path traverses offline switch s_i and is configured with the SDN routing mode, we have $y_i^l = 1$; otherwise $y_i^l = 0$. If flow f^l's forwarding path traverses online switch s_t and is configured with the legacy routing mode, we have $z_t^l = 1$; otherwise $z_t^l = 0$. The relationship between y_i^l, β_i^l, z_t^l, and ζ_t^l can be expressed as follows:

$$y_i^l \leq \beta_i^l, \quad \forall\, i,\ \forall\, l, \tag{1}$$

$$z_t^l \leq \zeta_t^l, \quad \forall\, t,\ \forall\, l. \tag{2}$$

In the above first inequality, the equal sign comes when f^l is forwarded under SDN mode at offline switch s_i. If the inequality sign is applied, f^l traverses s_i and is forwarded based on the legacy routing table without the controller. Similarly, as for the second inequality, the equal sign comes when f^l is forwarded under legacy mode at online switch s_t. If the inequality sign is applied, f^l traverses s_t and is forwarded based on the SDN routing table without the controller. The above two equations are feasible constraints. A flow can be configured at SDN mode at a switch only when (1) the flow traverses the switch and (2) the switch has at least two admissible paths to the flow's destination. If there is only one path of flow f: $s_1 \rightarrow s_2$ at switch s_1, we cannot change the path of flow at switch s_1, and flow does not possess any path programmability at switch s_1. As a result, we do not consider these flows in our paper. That is the reason why switches should have at least two paths to f^l's destination when β_i^l and z_t^l equal to 1. For simplicity, in the rest of this section, we use a switch instead of an offline switch.

B. Constraints

1) Switch-Controller Mapping Constraint: Each offline switch can be mapped to at most one controller. That is:

$$\sum_{j=1}^{M} x_{ij} \leq 1, \quad \forall\, i. \tag{3}$$

2) Controller Resource Constraint: When controllers fail, active controllers should try their best to control offline switches, which are previously controlled by failed controllers, without interrupting their normal operations. The control load of a controller equals to the total overhead of controlling the flows in its domain. We measure a controller's control resource

by the number of flows that the controller can normally control without introducing extra delays (*e.g.*, queueing delay [19]). However, we can release the available control resource of active controllers by configuring some flows at online switches under legacy mode. We use $A_j^{relaxed}$ to denote the released control resource after the configurations of flows at online switches under legacy mode and ν_{tj} to denote the relationship between active controller C_j and online switch s_t, that is:

$$\sum_{l=1}^{L}\sum_{t=N+1}^{T}(\nu_{tj} * \zeta_t^l * z_t^l) = A_j^{relaxed}, \quad \forall\ j.$$

The control load of a controller should not exceed the controller's available control resource. This can be written as follows:

$$\sum_{l=1}^{L}\sum_{i=1}^{N}(x_{ij} * \beta_i^l * y_i^l) \leq A_j^{rest} + A_j^{relaxed}, \quad \forall\ j, \tag{4}$$

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

3) Recovered Flow Programmability Constraint: If offline switch s_i on the path of flow f^l and uses SDN mode to route this flow, it can only change the next hop on the path of flow f^l. We use p_i^l to denote the number of paths from offline switch s_i's next hops to f^l's destination. Flow f^l's recovered path programmability at offline switch s_i is denoted as $rec\_pro_i^l$ and calculated as follows:

$$rec\_pro_i^l = \sum_{j=1}^{M}(x_{ij} * \beta_i^l * y_i^l * p_i^l), \quad \forall\ i,\ \forall\ l.$$

The recovered path programmability of flow f^l is denoted as rec_pro^l and equals to the sum of the path programmability at all offline switches. We use $\bar{p}_i^l$ to represent $\beta_i^l * p_i^l$, and rec_pro^l can be formulated as follows:

$$rec\_pro^l = \sum_{i=1}^{N} rec\_pro_i^l = \sum_{i=1}^{N}\sum_{j=1}^{M}(\bar{p}_i^l * x_{ij} * y_i^l), \quad \forall\ l.$$

We use p_t^l to denote the number of paths from online switch s_t's next hops to f^l's destination. Flow f^l's path programmability at online switch s_t is denoted as act_pro^l and calculated as follows:

$$act\_pro^l = \sum_{t=N+1}^{T}(\zeta_t^l - z_t^l) * p_t^l, \quad \forall\ l.$$

We use pro^l to denote the total programmability of flow f^l, and r to denote the least programmability of all offline flows. Thus, we have:

$$\begin{aligned} pro^l &= rec\_pro^l + act\_pro^l \\ &= \sum_{i=1}^{N}\sum_{j=1}^{M}(\bar{p}_i^l * x_{ij} * y_i^l) + \sum_{t=N+1}^{T}(\zeta_t^l - z_t^l) * p_t^l \\ &\geq r, \forall\ l. \end{aligned} \tag{5}$$

4) Propagation Delay Constraint: The propagation delay denotes the communication overhead between switches and controllers to control recovered flows, and it is a large share of the total communication overhead. The communication overhead between switches and controllers is one essential concern for WAN operators. High communication overhead may negatively affect several aspects (*e.g.*, routing policy updates, load balancing, and energy efficiency) of WANs and even degrade the network performance. Normally, to reduce the control propagation delay, switches are mapped to active controllers, which have enough control resource and are near to the switches. However, during controller failures, active controllers, which are close to offline switches, may not have enough control resource to recover offline switches. Simply mapping offline switches to their near active controllers could introduce extra processing delays [19] or even lead to cascading controller failure [9]. Thus, we set the propagation delay constraint to ensure that the offline switch is remapped to the active controller, which has enough control resource with a relatively low communication overhead. We use G to denote the total propagation delay of the ideal recovery case and is formulated as follows:

$$G = \sum_{j=1}^{M}\sum_{i=1}^{N}(\alpha_{ij} * \gamma_i * D_{ij}),$$

where α_{ij} denotes the relationship between offline switch s_i and controller C_j (*i.e.*, if controller C_j is the nearest controller of offline switch s_i, $\alpha_{ij} = 1$; otherwise $\alpha_{ij} = 0$), and γ_i denotes the number of flows in offline switch s_i.

To maintain low control propagation delay when recovering offline switches, we restrict the total propagation delay not exceeding the propagation delay of the ideal recovering case, where each offline switch is mapped to its nearest active controller without introducing extra delay. That is:

$$\sum_{l=1}^{L}\sum_{j=1}^{M}\sum_{i=1}^{N}(x_{ij} * \beta_i^l * y_i^l * D_{ij}) \leq G. \tag{6}$$

C. Objective Functions

The objective functions reflect our design considerations. Our problem has two objective functions. This first objective function reflects the first and second considerations and aims to maximize the number of recovered offline flows and let each flow have similar programmability. That is:

$$obj_1 = r.$$

The second objective function reflects the third consideration and is to make full use of the control resource of active controllers by improving the total programmability of all flows switches. That is:

$$obj_2 = \sum_{l=1}^{L} pro^l.$$

D. Problem Formulation

The goal of our problem is to recover offline flows with the similar path programmability and maximize the total programmability of all flows at by selecting the routing mode of flows at recovered switches and smartly mapping the recovered switches to active controllers. The control resource of active controllers can be released by configuring some normal flows under legacy routing mode.

Typically, we have two options to formulate the JFMSSM problem. The first option is to formulate a two-stage problem. In practice, the first objective function has the highest priority because it maintains the essential of SDN. Thus, the first-stage problem tries to recover each offline flow to have the similar path programmability. The second-stage problem maximizes the total programmability based on the path programmability of flows obtained by solving the first-stage problem. The second option is to formulate one problem by combining the two objective functions into one objective function. Compared with the two-stage problem formulation that needs to solve two problems, the second option only solves one problem. In this problem, we select the second option. We formulate the objective function as follows:

$$obj = obj_1 + \lambda * obj_2 = r + \lambda \sum_{l=1}^{L} pro^l,$$

where λ ($\lambda \geq 0$) is a constant number gives different weights of the two objective functions. Therefore, we can formulate the problem as follows:

$$\begin{aligned} &\max_{r,x,y,z} \ r + \lambda \sum_{l=1}^{L} pro^l \\ &\text{s.t. } (1)(2)(3)(4)(5)(6), \\ &\quad r \geq 0, x_{ij}, y_i^l, z_t^l \in \{0,1\}, \quad \forall\ i,\ \forall\ j,\ \forall\ l,\ \forall\ t. \end{aligned} \tag{P}$$

E. Problem Reformulation

In the JFMSSM problem, the objective function is linear, variables are binary integers, and constraints are nonlinear. Thus, this problem is a Mixed-Integer Nonlinear Programming (MINLP), which is widely known for the NP-hard computation complexity. One complexity of the JFMSSM problem comes from Eqs. (4), (5), and (6) since two binary variables are multiplied in the two equations. To efficiently solve the problem, we reformulate the problem to a Mixed-Integer Linear Programming (MILP) by equivalently linearizing the bilinear terms of binary variables $x_{ij} * y_i^l$ in Eqs. (4), (5), and (6). Specifically, we can replace the bilinear term $x_{ij} * y_i^l$ by introducing an auxiliary binary variable ω_{ij}^l and add the following linear constraints:

$$x_{ij} \geq \omega_{ij}^l, \quad \forall\ i,\ \forall\ j,\ \forall\ l. \tag{7}$$

$$y_i^l \geq \omega_{ij}^l, \quad \forall\ i,\ \forall\ j,\ \forall\ l. \tag{8}$$

$$(x_{ij} + y_i^l - 1) \leq \omega_{ij}^l, \quad \forall\ i,\ \forall\ j,\ \forall\ l. \tag{9}$$

TABLE II
NOTATIONS

Notation	Meaning
$\mathcal{S}$	the set of offline switches, $\mathcal{S} = \{s_i \mid i \in [1, N]\}$
$\mathcal{S}'$	the set of online switches, $\mathcal{S}' = \{s_t \mid i \in [N+1, T]\}$
$\mathcal{C}$	the set of active controllers, $\mathcal{C} = \{C_j \mid j \in [1, M]\}$
$\mathcal{F}$	the set of flows, $\mathcal{F} = \{f^l \mid l \in [1, L]\}$
x_{ij}	a binary design variable that denotes if offline switch s_i is controlled by controller C_j
y_{il}	a binary design variable that denotes if flow f^l is configured under SDN mode at offline switch s_i
z_{tl}	a binary design variable that denotes if flow f^l is configured under legacy mode at online switch s_t
β_{il}	a binary constant that denotes if flow f^l traverses offline switch s_i
ζ_{il}	a binary constant that denotes if flow f^l traverses online switch s_t
$\mathcal{D}(i)$	the propagation delay of offline switch s_i to active controllers, $\mathcal{D}(i) = \{D_{i1}, ..., D_{ij}, ..., D_{iM}\}, i \in [1, N]$
$\mathcal{C}(i)$	the set of active controllers by sorting $\mathcal{C} = \{C_j \mid j \in [1, M]\}$ following the ascending order of $\mathcal{D}(i)$, $i \in [1, N]$
$\mathcal{A}$	the set of the available processing capacity of controllers, $\mathcal{A} = \{A_j^{rest} \mid j \in [1, M]\}$, where A_j^{rest} denotes the available processing capacity of controller C_j
$\mathcal{H}$	the set of temporary path programmability of flows, $\mathcal{H} = \{h^l \mid l \in [1, L]\}$, where h^l denotes the temporary path programmability of flow f^l
rec_pro^l	the recovered path programmability of flow f^l, which equals to the sum of the recovered path programmability at all offline switches
act_pro^l	the online path programmability of flow f^l, which equals to the sum of the path programmability at all online switches
$\mathcal{P}$	the set of the path programmability, $\mathcal{P} = \{pro^l \mid l \in [1, L]\}$, where pro^l denotes the path programmability of flow f^l, and is formulated as the sum of recovered path programmability rec_pro^l and normal path programmability act_pro^l
$\mathcal{X}$	the set of the mapping relationship between offline switches and active controllers, $\mathcal{X} = \{(i,j) \mid i \in [1, N], j \in [1, M]\}$
$\mathcal{Y}$	the set of the mode setting relationship between offline flows and offline switches, $\mathcal{Y} = \{(i,l) \mid i \in [1, N], l \in [1, L]\}$
σ	the auxiliary programmability variable that indicates the least programmability of all offline flows
δ	the auxiliary flow variable that indicates the number of offline flows that have the least programmability in all offline switches
Γ	the set of the number of flows at offline switches, $\Gamma = \{\gamma_i \mid j \in [1, M]\}$, where γ_i denotes the number of flows traversing s_i
μ	the constant number that indicates the degree of releasing the control resource of active controllers. We set μ to be 0.5 in this paper
TOTAL_ITERATIONS	the maximum number of offline switches in the original path of offline flows

Thus, Eqs. (4), (5), and (6) can be respectively reformulated as follows:

$$\sum_{l=1}^{L} \sum_{i=1}^{N} (\omega_{ij}^l * \beta_i^l) \leq A_j^{rest} + A_j^{relaxed}, \quad \forall\ j. \tag{10}$$

$$pro^l = \sum_{i=1}^{N} \sum_{j=1}^{M} (\bar{p}_i^l * \omega_{ij}^l) + \sum_{t=N+1}^{T} (\zeta_t^l - z_t^l) * p_t^l \geq r, \quad \forall\ l. \tag{11}$$

$$\sum_{l=1}^{L} \sum_{j=1}^{M} \sum_{i=1}^{N} (\omega_{ij}^l * \beta_i^l * D_{ij}) \leq G. \tag{12}$$

Therefore, the JFMSSM problem can be reformulated as follows:

$$\max_{r,x,y,z,\omega} \; r + \lambda \sum_{l=1}^{L} pro^l$$
$$\text{s.t. } (1)(2)(3)(7)(8)(9)(10)(11)(12),$$
$$r \geq 0, x_{ij}, y_i^l, z_t^l, \omega_{ij}^l \in \{0,1\}, \quad \forall\, i,\ \forall\, j,\ \forall\, l,\ \forall\, t. \tag{P'}$$

V. FlexPM Design

The typical solution to the JFMSSM problem is to get its optimal result using existing IP optimization solvers. However, as the network size increases, the solution space could increase significantly, and finding a feasible solution could cost a long time or perhaps is impossible. We design the heuristic algorithm named FlexPM to solve the problem and achieve the trade-off between the performance and time complexity.

The key idea of FlexPM includes two parts: (1) preferably recovering offline flows, which have the least programmability and (2) making full use of available control resource to improve the total programmability. Details are summarized in Algorithm 1, and the notations used in the algorithm are listed in Table II. At the beginning of the algorithm, in line 1, we initialize $\mathcal{X}$ and $\mathcal{Y}$ to be empty, set test switch-set $\mathcal{S}^*$ to $\mathcal{S}$, auxiliary programmability variable σ to 0, tested_counter to 0, and get the new set of the available control resource $\mathcal{A}$ after the release of control resource by configuring normal flows under legacy routing mode. In line 2, we start iteratively to find the feasible mappings between switches and controllers and select the routing mode for offline flows at switches. During an iteration, the programmability of flows, which have the least programmability, can be increased by one at most to balance the path programmability of each flow. To guarantee that all the offline switches in the original path of each flow are tested, the program totally runs TOTAL_ITERATIONS iterations. If the program continues after TOTAL_ITERATIONS times iterations, the path programmability of offline flows will not get improved because all possible improvements for each offline flow are already tested. In line 3, for each iteration, we set auxiliary flow variable δ to 0, the index of recovering switch i_0 to NULL, and the index of mapping controller j_0 to NULL. In lines 5-15, we select switch s_{i_0} to recover. First, we collect the number of flows, which have the least programmability from $\mathcal{S}^*$ (lines 6-11), and select the switch, which has the maximum number of flows with the least programmability. Then, we choose switch s_{i_0} to recover by updating this switch's index to i_0 (lines 12-14).

Lines 17-29 map switch s_{i_0} to controller C_{j_0}. In line 18, if switch s_{i_0} is already remapped, we use index i_0 to find index its controller's index j_0 in $\mathcal{X}$. Otherwise, switch s_{i_0} is not remapped to any active controller, and we find an active controller in lines 20-27. In lines 20-24, we test controllers following the ascending order of the propagation delay with switch s_{i_0}. If there exists a controller that has enough control resource to control switch s_{i_0}, we establish the mapping

Algorithm 1 FlexPM

Input: $\mathcal{S}$, $\mathcal{C}$, $\mathcal{A}$, $\mathcal{D}$, $\mathcal{P}$, $\mathcal{R}$, TOTAL_ITERATIONS, μ;
Output: $\mathcal{X}$, $\mathcal{Y}$;

1: $\mathcal{X} = \emptyset$, $\mathcal{Y} = \emptyset$, $\mathcal{S}^* = \mathcal{S}$, $\sigma = 0$, test_count = 0, $\mathcal{A}$ = FlowModeSelection($\mathcal{A}$, $\mathcal{C}$, μ);
2: **while** test_count $\leq$ TOTAL_ITERATIONS **do**
3: δ = 0, i_0 = NULL, j_0 = NULL;
4: //find switch s_{i_0} to recover
5: **for** $s_i \in \mathcal{S}^*$ **do**
6: TEST_NUM $= 0$;
7: **for** $l \in \{\beta_i^l = 1, l \in [1, L]\}$ **do**
8: **if** $h^l == \sigma$ **then**
9: TEST_NUM = TEST_NUM + 1;
10: **end if**
11: **end for**
12: **if** TEST_NUM $> \delta$ **then**
13: $\delta =$ TEST_NUM, $i_0 = i$;
14: **end if**
15: **end for**
16: //map switch s_{i_0} to controller C_{j_0}
17: **if** $(i_0, *) \in \mathcal{X}$ **then**
18: use i_0 to find j_0 from $\mathcal{X}$;
19: **else**
20: **for** $C_j \in C(i_0)$ **do**
21: **if** $A_j^{rest} \geq \gamma_{i_0}$ **then**
22: $j_0 = j$;
23: **end if**
24: **end for**
25: **if** $j_0 ==$ NULL **then**
26: A_j^{rest} = max($\mathcal{A}$), $j_0 = j$;
27: **end if**
28: **end if**
29: $\mathcal{X} \leftarrow \mathcal{X} \cup (i_0, j_0)$, $\mathcal{S}^* \leftarrow \mathcal{S}^* \setminus s_{i_0}$;
30: //select the routing mode for flows at switch s_{i_0}
31: **for** $l_0 \in \{\beta_{i_0}^l = 1, l \in [1, L]\}$ **do**
32: **if** $h^{l_0} \leq \sigma$ and $(i_0, l_0) \notin \mathcal{Y}$ and $A_{j_0}^{rest} > 0$ **then**
33: $A_{j_0}^{rest} = A_{j_0}^{rest} - 1$, $h^{l_0} = h^{l_0} + pro_{i_0}^{l_0}$;
34: $\mathcal{Y} \leftarrow \mathcal{Y} \cup (i_0, l_0)$;
35: **end if**
36: **end for**
37: **if** $|\mathcal{S}^*| == \emptyset$ **then**
38: $\mathcal{S}^* = \mathcal{S}$, test_count++, σ = min($\mathcal{H}$);
39: **end if**
40: **end while**
41: //improve the total programmability
42: **for** $(i_0, l_0) \in \{\beta_i^l = 1, i \in [1, M], l \in [1, L]\}$ **do**
43: **if** $(i_0, *) \in \mathcal{X}$ **then**
44: use i_0 to find j_0 from $\mathcal{X}$;
45: **if** $A_{j_0}^{rest} > 0$ and $(i_0, l_0) \notin \mathcal{Y}$ **then**
46: $A_{j_0}^{rest} = A_{j_0}^{rest} - 1$, $h^{l_0} = h^{l_0} + pro_{i_0}^{l_0}$;
47: $\mathcal{Y} \leftarrow \mathcal{Y} \cup (i_0, l_0)$;
48: **end if**
49: **end if**
50: **end for**
51: return $\mathcal{X}, \mathcal{Y}$;

Algorithm 2 FlowModeSelection()

Input: $\mathcal{A}$, $\mathcal{C}$, μ;
Output: $\mathcal{A}$;

1: Generate $\bar{Z} = \{z_k, k \in [1, L*(T-N)]\}$ by solving the linear programming relaxation of problem (P') and sorting the results in the descending order;
2: **for** $z_k \in \bar{Z}$ **do**
3: **if** $z_k < \mu$ **then**
4: //the flow mode selection procedure for normal flows which aims to release control resource stops;
5: break;
6: **end if**
7: get the switch and flow IDs t_0 and l_0 of z_k, and find the active controller C_{j_0} that controls the switch s_{t_0} from set $\mathcal{C}$;
8: //update $A_{j_0}^{rest}$;
9: $A_{j_0}^{rest} = A_{j_0}^{rest} + 1$;
10: **end for**
11: return $\mathcal{A}$;

between switch s_{i_0} and controller C_{j_0}. If we cannot find a capable controller, in line 26, we find C_{j_0}, the controller with the maximum available control resource, to control switch s_{i_0}. In line 29, the mapping between switch s_{i_0} and controller C_{j_0} is established, set $\mathcal{X}$ is updated according, and the mapped switch s_{i_0} is removed from $\mathcal{S}^*$.

Lines 31-36 select routing mode for offline flows in switch s_{i_0}. We try to select offline flows with the least programmability and configure them working under the SDN mode without overloading controller C_{j_0}. C_{j_0}'s control resource A_j^{rest}, recovered flow f^{l_0}'s routing mode and programmability h^{l_0} are updated accordingly.

In lines 37-39, if all switches are tested, we should start a new iteration. This is because the least programmability could be further improved by configuring more flows working under SDN mode at switches. To restart the iteration, we configure $\mathcal{S}^*$ to $\mathcal{S}$, increase tested_counter by one, and upgrade the least programmability σ.

After the iteration from lines 2 to 40, the least programmability cannot be further increased. However, active controllers' control resource may not be fully utilized, and there is still room to improve the total programmability. As a result, in lines 42-50, we test all flows to find out if there still exists feasible SDN mode configurations for flows to further improve the total programmability. In line 51, the routing mode of recovered flows at recovered switches and switch-controller mappings are finally generated.

Algorithm 2 shows details of function FlowModeSelection(). The general idea of this function is to relax the original problem into a linear programming problem and then finely round the solution of the linear programming problem to get the final result of the original problem. The function can select proper normal flows and configure them under legacy routing mode to release the control resource of active controller.

VI. Simulation

A. Simulation Setup

We use two typical backbone topology AT&T and Belnet from Topology Zoo [20] to evaluate the performance of FlexPM. AT&T is one of the world's largest Internet service providers, and it has started to softwarize its WAN with programmable devices. The AT&T topology is the national primary topology of US and consists of 25 nodes and 112 links. The Belnet topology is a national topology of Belgium and consists of 21 nodes and 48 links. Some existing works [7], [8], [14], [21], [22] also study the problem of maintaining network programmability in WAN by using AT&T and Belnet topologies as the simulation topologies. Thus, following existing works, we also use these two topologies as representative WAN topologies. In both AT&T and Belnet topologies, each node has a unique ID, a latitude, and a longitude. We calculate the distance between two nodes using Haversine formula [23] and use the distance divided by the propagation speed (*i.e.*, 2×10^8 m/s) [24] to represent the propagation delay between the two nodes. In our simulation, each node is an SDN switch, and any two nodes have a traffic flow forwarded on the shortest path. Following existing works [7], [10], [11], the control plane consists of six controllers under AT&T topology and five controllers under Belnet topology, and the processing ability of each controller is 500. Table III and V show the default relationship of controllers, switches, and the number of flows in the switches under the AT&T topology and Belnet topology.

B. Comparison Algorithms

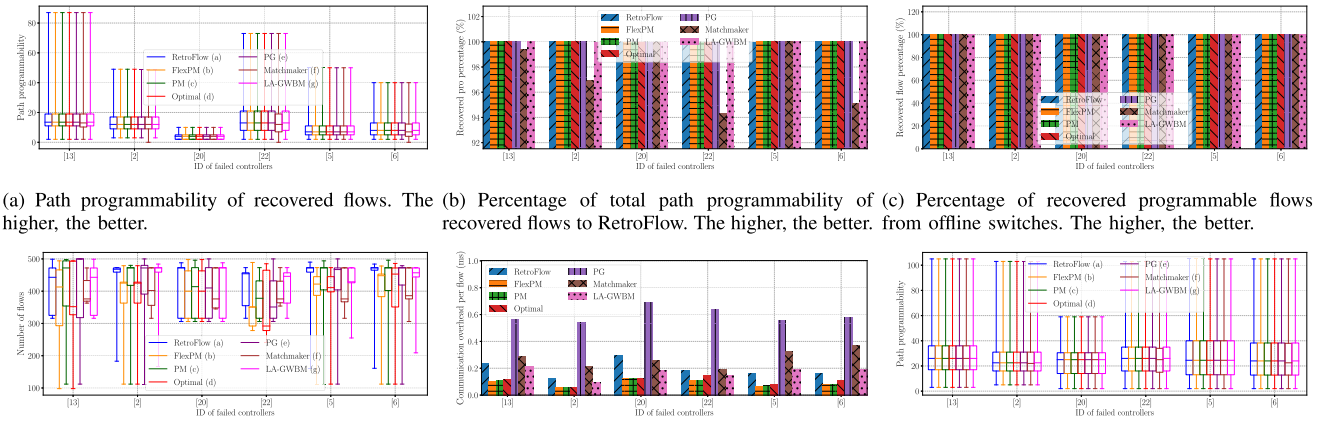
1) Optimal: it is the optimal solution to problem P'. We solve it with an advanced solver GUROBI [25].
2) RetroFlow [10]: this solution recovers offline flows by configuring a set of offline switches working under the legacy routing mode and transferring the control of offline switches with the SDN routing mode to active controllers.
3) ProgrammabilityGuardian (PG) [11]: this solution realizes fine-grained flow-controller mappings using a middle layer between controllers and switches.
4) ProgrammabilityMedic (PM) [1]: this solution recovers offline flows with the similar path programmability and maximizes the total programmability of recovered flows by deciding the proper mode of flows at recovered offline switches and configuring switch-controller mappings.
5) FlexPM: this algorithm is shown in Algorithm 1.
6) Matchmaker [7]: this algorithm adaptively adjusts the control cost of offline switches based on the limited control resource by changing the paths of flows to realize proper offline switches remapping.
7) LA-GWBM [8]: this is a greedy algorithm to assign master and slave controllers to switches against multiple controller failures.

C. Simulation Results

We compare the performance of FlexibleProgrammabilityMedic with other algorithms under three scenarios with

TABLE III
DEFAULT RELATIONSHIP BETWEEN CONTROLLERS, SWITCHES, AND THE NUMBER OF FLOWS IN THE SWITCHES UNDER AT&T 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) Path programmability of recovered flows. The higher, the better.

(b) Percentage of total path programmability of recovered flows to RetroFlow. The higher, the better.

(c) Percentage of recovered programmable flows from offline switches. The higher, the better.

(d) Control resource of active controllers.

(e) Per-flow communication overhead. The lower, the better.

(f) Overall performance of all flows. The higher, the better.

Fig. 4. Results of one controller failure under AT&T topology. (a) to (g) are used to denote the seven solutions from left to right in each scenario.

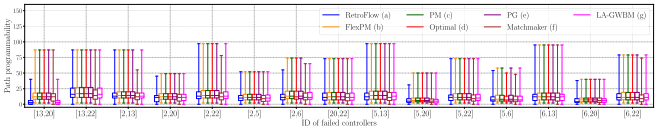
two topologies: one controller failure, two controller failures and three controller failures. We also evaluate FlexPM's computation efficiency. Following our two objective functions, we use two metrics: path programmability of recovered flows, which reflects the performance of obj_1, and the overall performance of all flows, which reflects the performance of obj_2.

1) Under AT&T Topology: ***One controller failure:*** Fig. 4 shows the results of seven algorithms when one of six controllers fails. One controller failure is a common scenario, and there are 6 combinations. In all six cases, all algorithms recover 100% offline switches, and active controllers usually have enough control resource to recover all offline switches, as shown in Fig. 4(d). In Fig. 4(a), all algorithms realize the same programmability of recovered flows except for Matchmaker. In Fig. 4(b), the rest of the six solutions exhibit the same total programmability of recovered flows since all algorithms recover 100% offline flows, as shown in Fig. 4(c). Matchmaker's performance is slightly lower than the other algorithms because it does not take the programmability into consideration while recovering the offline flows. Matchmaker only aims to maximize the number of recovered offline flows, which is also shown as Fig. 4(c). Fig. 4(e) shows the per-flow communication overhead of all algorithms. The total communication overhead comes from the propagation delay due to the geographical distance, and the overhead is the total communication overhead divided by the number of recovered flows. In this figure, FlexPM requires the least overhead because it takes the propagation delay into consideration. While PG performs worst since it introduces a middle layer using FlowVisor, which needs 0.48 ms on average to pull for port status [12]. In Fig. 4(f), we can see that Matchmaker's overall performance (*i.e.*, path programmability of all flows) is marginally lower than the rest of the six schemes, because it does not realize 100% path programmability recovery.

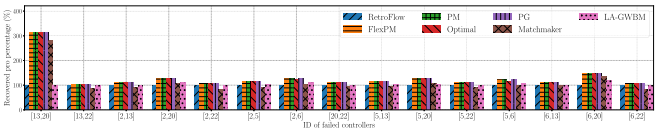
Two controller failures: Fig. 5 shows the results of seven algorithms when two of six controllers fail. Two controllers failure is a moderate failure scenario, and there are 15 combinations.

(1) Path programmability of recovered flows: Fig. 5(a) shows the path programmability of recovered flows. For all cases, the least path programmability of RetroFlow, Matchmaker, and LA-GWBM are all 0 since some offline flows are not recovered, and their medians are always lower than FlexPM, Optimal, PM, and PG. That is because theses three schemes can only realize per-switch routing granularity, which promises mediocre recovery performance.

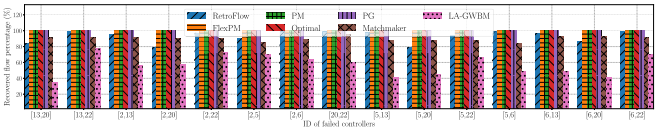
In this figure, some flows have high programmability, but the least path programmability is limited to 2. The high programmability comes from the flows with long paths, and the least programmability is constrained by specific flows with short paths. In our simulation, we generate a flow for any two nodes. In each of the 15 cases, we can always find two types of offline flows: (1) a flow's source and destination nodes are directly connected; (2) a flow's shortest path only includes one node except its source and destination nodes, and this only one node just has two paths to the flow's destination node. Either of the two types of flows' programmability is limited to 2, and thus the performance of FlexPM, Optimal, PM, and PG are limited by these flows. However, the results still indicate that FlexPM, Optimal, PM, and PG maintain balanced programmability because all the flows are recovered to have the programmability of 2 at least. This figure also proves that our second objective of maximizing the total programmability is critical because it can improve the utilization of control resource in active controllers for flow recovery. Without this



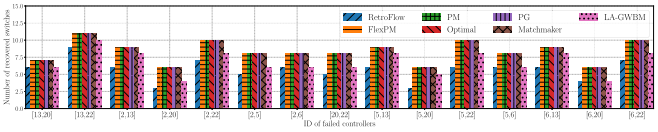
(a) Path programmability of recovered flows. The higher, the better.



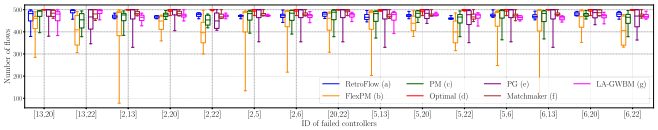
(b) Percentage of total path programmability of recovered flows to RetroFlow. The higher, the better.



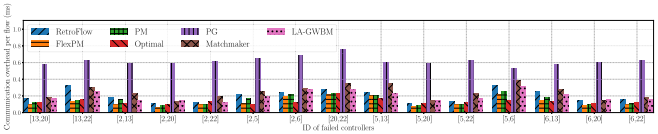
(c) Percentage of recovered programmable flows from offline switches. The higher, the better.



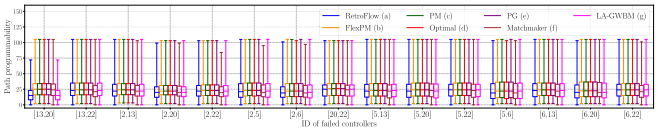
(d) Number of recovered offline switches. The higher, the better.



(e) Control resource of active controllers.



(f) Per-flow communication overhead. The lower, the better.



(g) Overall performance of all flows. The higher, the better.

Fig. 5. Results of two controller failures under AT&T topology. (a) to (g) are used to denote the seven solutions from left to right in each scenario.

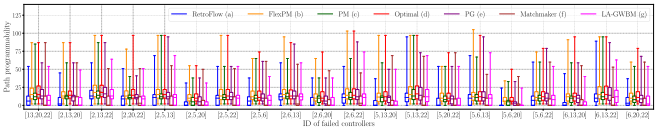
objective, all flows will just be recovered with the least programmability.

(2) Total path programmability of recovered flows: Fig. 5(b) shows the total path programmability. To clearly show the performance difference, we normalize the result of each algorithm to the result of RetroFlow. FlexPM, PM, and PG perform best among the seven algorithms, and FlexPM improves the performance from 105% to 315% because FlexPM recovers more offline flows. Figs. 5(c), (d), and (e) show the percentage of recovered flows, the percentage of recovered switches, and the control resource of active controllers, respectively. LA-GWBM performs worse than others since it recovers a small number of offline flows. RetroFlow and Matchmaker perform slightly better than LA-GWBM, but both of them cannot achieve 100% recovery performance in any scenario. FlexPM, Optimal, PM, and PG perform the same in offline flows recovery. For all cases, RetroFlow only recovers flows in the range of 71% to 99% and only recovers part of offline switches because the coarse-grained switch-controller mappings cannot fully occupy active controllers to recover flows without overloading active controllers. If an offline flow only traverses unrecovered switches, it remains offline.

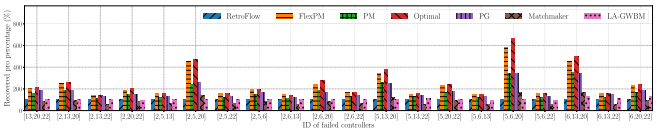
In contrast, PG recovers all the offline switches thanks to the fine-grained flow-controller mappings. By employing the hybrid routing mode, Optimal and FlexPM, and PM approximately realize fine-grained flow-level recovery. Thus, they can change the control cost of offline switches and enable active controllers to recover more switches than existing switch-level solutions, Thus, they increase the number of recovered switches and achieve the performance same to PG to recover all flows. For the special failure case (13, 20), FlexPM's performance is 315% of RetroFlow's. Specifically, with RetroFlow, 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 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, FlexPM can recover the programmability in a fine-grained per-flow mode by dynamically altering the control cost based on the given control resource. Supported by high-end commercial SDN switches with hybrid routing mode, 188 of 213 flows are configured with SDN routing mode at switch s_{13}, and the rest of the flows are with legacy mode. So, FlexPM 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.

(3) Communication overhead: Our problem formulation uses Eq. (12) to limit the overall communication overhead. Because each algorithm recovers different number of flows, both the number of recovered flows and overall communication overhead should be considered in the evaluation metric. Fig. 5(f) shows the per-flow communication overhead. For all cases, PG performs worst and is about three to four times higher than FlexPM on average. This is because the middle layer's processing time significantly increases the overhead. FlexPM outperforms other six algorithms in the majority of cases, and FlexPM's performance is lower than Optimal's in 10 of 15 cases. That is because the total propagation delay G under the ideal recovery case varies under different failure cases. In the 10 of 15 cases, the total propagation delay of heuristic FlexPM is lower than G, but the total propagation delay of Optimal can be only limited to G.

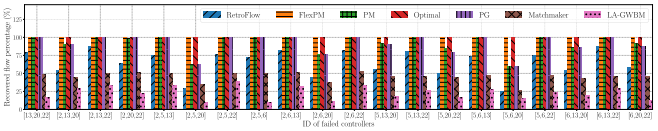
(4) Overall performance of all flows: Fig. 5(g) shows the overall performance of all flows. FlexPM, Optimal, PM, and PG perform the best because they recovered all path programmbility from offline flows, as shown in Fig. 5(a), (b), and (c). FlexPM has sufficient available control resource to recover all offline flows, thus FlexPM is not required to release any control resource of normal flows. The overall performance of three per-switch mapping solutions (*i.e.*,



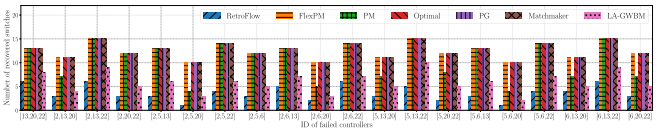
(a) Path programmability of recovered flows. The higher, the better.



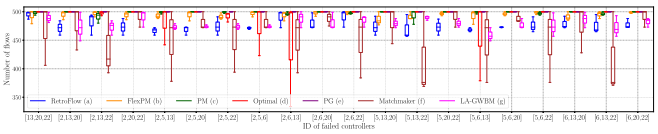
(b) Percentage of total path programmability of recovered flows to RetroFlow. The higher, the better.



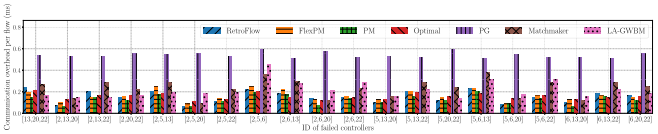
(c) Percentage of recovered programmable flows from offline switches. The higher, the better.



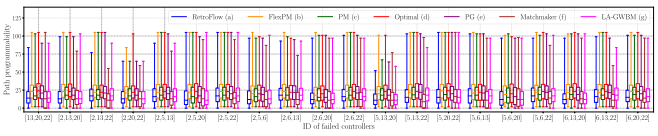
(d) Number of recovered offline switches. The higher, the better.



(e) Control resource of active controllers. Closer to 500 is better.



(f) Per-flow communication overhead. The lower, the better.



(g) Overall performance of all flows. The higher, the better.

Fig. 6. Results of three controller failures under AT&T topology. (a) to (g) are used to denote the seven solutions from left to right in each scenario.

RetroFlow, Matchmaker, and LA-GWBM) are not comparable with the performance of four per-flow mapping solutions.

Three controller failures: Fig. 6 shows the results of seven algorithms when three of six controllers fail. Three controllers failure is a serious failure scenario, and there are 20 combinations.

(1) Path programmability of recovered flows: Fig. 6(a) shows the path programmability of recovered flows. Similar to the two controller failures, for all cases, RetroFlow's, Matchmaker's, and LA-GWBM's least path programmability are all 0 because some flows are not recovered, and their medians are always lower than FlexPM, Optimal, PM, and PG. However, the performance of four per-flow mapping schemes vary significantly. It can be seen that both PM and PG cannot recover all offline flows in some cases, and their medians are always lower than FlexPM and Optimal. FlexPM and Optimal exhibit balanced programmability because the least programmability is recovered to 2 in all cases. That is because PM and PG have limited control resource to recover all offline flows. On the contrary, FlexPM and Optimal can carefully select the proper normal flows and configure them under legacy mode at online switches to further release control resource for better offline flows recovery performance.

TABLE IV

PERCENTAGE OF COMPUTATION TIME OF FLEXPM TO OPTIMAL UNDER AT&T TOPOLOGY. THE LOWER, THE BETTER

Scenario	One controller failure (%)	Two controller failures (%)	Three controller failures (%)
FlexPM	61.96	31.53	22.50
Optimal	100.00	100.00	100.00

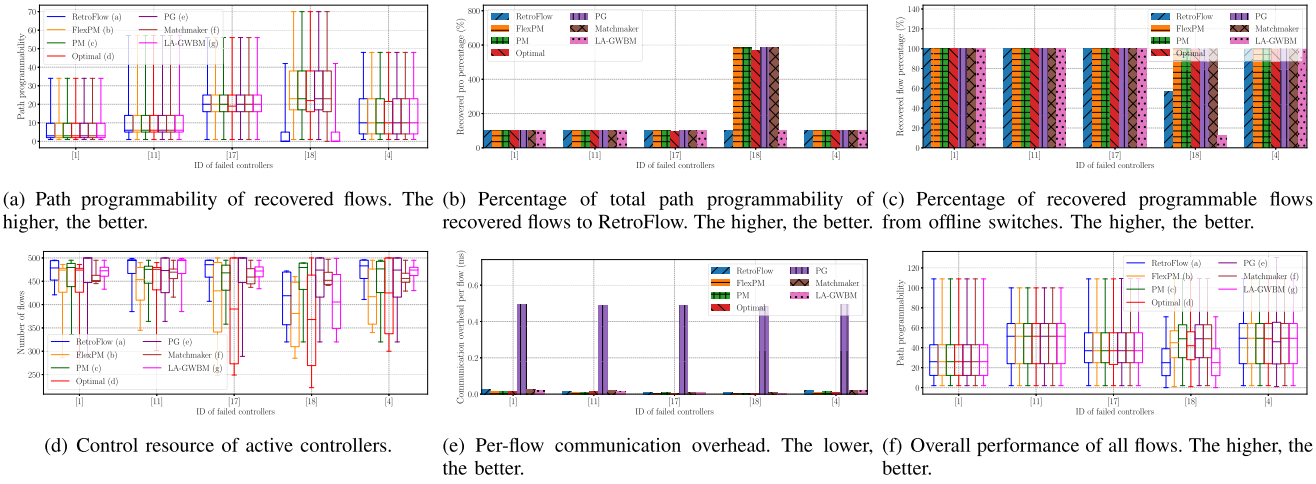
(2) Total path programmability of recovered flows: Fig. 6(b) shows the total path programmability. For the 20 cases, Optimal performs best in 19 cases. For case (2, 6, 13), FlexPM's performance is better than Optimal since the performance of Optimal is restricted by the propagation delay constraint, as shown in Fig. 6(f). FlexPM performs closer to Optimal and better than other five schemes. For failure case (5, 6, 20), FlexPM's performance is 320% higher than PM's and 660% of RetroFlow's. The control cost of offline switch s_{13} is 223, but the available control resource of active controllers C_2, C_5, and C_{22} are 124, 184, and 34, respectively. Neither of the active controllers can re-control switch s_{13} due to the per-switch mapping. On the contrary, FlexPM can recover the programmability in a fine-grained per-flow mode by dynamically altering the control cost of offline switches based on given control resource of active controllers, and FlexPM also releases some control resource of active controller to recover more offline flows.

Per-switch mapping solutions (*i.e.*, RetroFlow, Matchmaker, and LA-GWBM) perform worse than others since they perform worse in path programmability and the percentage of recovered flows. Fig. 6(c) shows the percentage of recovered flows. For all cases, RetroFlow only recovers flows in the range of 25% to 85%. PM and PG realize 100% flow recovery in 11 and 12 of 20 cases, respectively. PM's performance is comparable with PG by recovering flows in the range of 60% to 92%. Both FlexPM and Optimal can realize 100% offline flows recovery in all cases since they have more available control resource by configuring some normal flows under legacy routing mode at online switches. Fig. 6(d) shows the percentage of recovered switches. For all cases, RetroFlow and LA-GWBM can only recover a small part of offline switches because the switch-level mapping solution cannot fully utilize the available resource of active controllers, as shown in Fig. 6(e).

(3) Communication overhead: Fig. 6(f) shows per-flow communication overhead. PG performs worst for all cases while FlexPM performs best in most of the cases. Note that in some cases, RetroFlow slightly outperforms FlexPM because

TABLE V
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



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

(b) Percentage of total path programmability of recovered flows to RetroFlow. The higher, the better.

(c) Percentage of recovered programmable flows from offline switches. The higher, the better.

(d) Control resource of active controllers.

(e) Per-flow communication overhead. The lower, the better.

(f) Overall performance of all flows. The higher, the better.

Fig. 7. Results of one controller failure under Belnet topology. (a) to (g) are used to denote the seven solutions from left to right in each scenario.

the path programmability of each recovered flow of RetroFlow is far less than FlexPM, as shown in Fig. 6(a).

(4) Overall performance of all flows: Fig. 6(g) shows the overall performance of all flows. FlexPM and Optimal perform the best since they enjoy the higher least path programmability and median path programmability. It can be seen from this figure that the release of control resource by configuring normal flows under legacy mode has little impact on the path programmability of these normal flows, since FlexPM can reasonably degrade the normal flows with high path programmability to recover more offline flows and maximize the overall performance of all flows. As a result, FlexPM can guarantee more balanced path programmability of flows than PM. Three per-switch mapping solutions (*i.e.*, RetroFlow, Matchmaker, and LA-GWBM) perform the worst because they recover less offline flows.

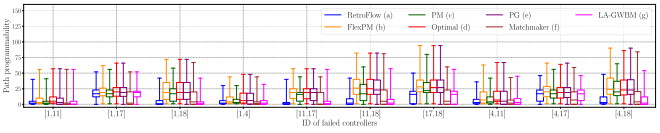
Computation time: To show the computation efficiency, we evaluate the computation time of FlexPM and Optimal under the above three scenarios, and Table IV shows the result. In the figure, the computation time of FlexPM is only 61.96%, 31.53%, and 22.50% of Optimal on average under three failure scenarios. Thus, considering recovery performance in Figs. 4, 5, and 6, the result indicates that the proposed FlexPM can achieve good recovery performance with low computation time. It can be also seen that the computation complexity of Optimal increases significantly as the controller failure scenario gets severer.

2) Under Belnet Topology: ***One controller failure:*** Fig. 7 shows the results of seven algorithms when one of six controllers fails. There are 5 combinations of one controller failure scenario. In Fig. 7(a), all seven solutions recover all offline flows except the failure case (18). For the special failure case (18), the control cost of switch s_{18} 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, RetroFlow, Matchmaker, and LA-GWBM cannot fully recover switch s_{18}. Matchmaker outperforms RetroFlow and LA-GWBM by altering the control cost of switch s_{18} based on the control resource of active controllers but cannot realize 100% recovery of offline flows, as shown in Fig. 7(c). On the contrary, per-flow mapping solutions (*i.e.*, FlexPM, Optimal, PM, and PG) can recover all offline flows with high path programmability. FlexPM's performance of recovered total path programmability is 590% of RetroFlow's.

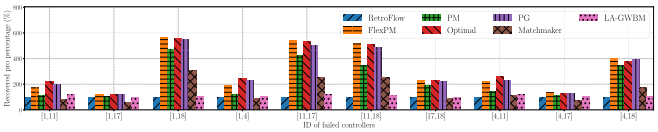
Figs. 7 (e) and (f) show the control resource of active controllers and the per-flow communication overhead, respectively. Recall that the extra processing delays introduced by FlowVisor and the propagation delay from distance variation. In the two topologies, the extra processing delay contributes more than the propagation delay in the overhead. However, in Belnet topology, the propagation delay of the seven algorithms are significantly lower than in AT&T topology, since the Belnet topology is much smaller than the AT&T topology. Except for PG, the rest of the six schemes do not introduce extra processing delays. In Fig. 7(f), the overall performance of RetroFlow and LA-GWBM are dramatically worse than the rest of the five schemes, since they can only recover about 58% and 12% offline flows.

Two controller failures: Fig. 5 shows the results of seven algorithms when two of six controllers fail. There are 10 combinations of two controller failures scenario.

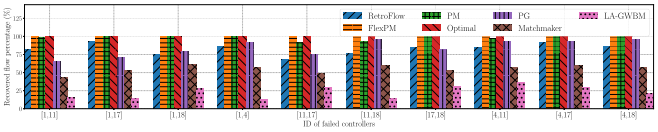
(1) Path programmability of recovered flows: Fig. 8(a) shows the path programmability of recovered flows. Only FlexPM and Optimal exhibit 100% offline flows recovery with the least programmability of 2 in all cases because they smartly release some control resource of active controllers which are used to control the normal flows. On the contrary, the other two per-flow mapping solutions (*i.e.*, PM and PG)



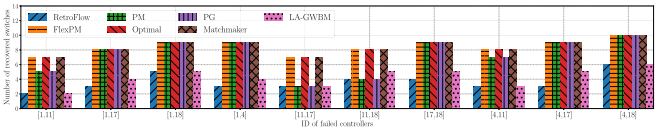
(a) Path programmability of recovered flows. The higher, the better.



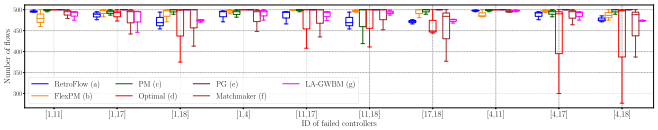
(b) Percentage of total path programmability of recovered flows to RetroFlow. The higher, the better.



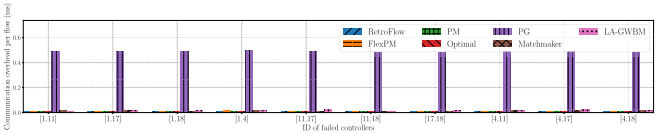
(c) Percentage of recovered programmable flows from offline switches. The higher, the better.



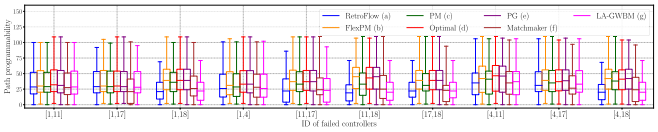
(d) Number of recovered offline switches. The higher, the better.



(e) Control resource of active controllers.



(f) Per-flow communication overhead. The lower, the better.



(g) Overall performance of all flows. The higher, the better.

Fig. 8. Results of two controller failures under Belnet topology. (a) to (g) are used to denote the seven solutions from left to right in each scenario.

cannot recover all offline flows in 4 of 10 cases, since they have limited available control resource. Per-switch mapping solutions perform the worst.

(2) Total path programmability of recovered flows: Fig. 8(b) shows the total path programmability. FlexPM and Optimal perform the best since they have sufficient control resource to recover all offline flows.For failure case (11, 18), FlexPM's performance is 210% higher than PM and 550% of RetroFlow. Note that FlexPM outperforms Optimal in several cases, and Optimal has ample control resource as shown in Fig. 8(e), that is because Optimal is restricted by the processing delay constraint.

(3) Communication overhead: Fig. 8(f) shows the per-flow communication overhead. Similar to Fig. 7(e), PG performs the worst since it experiences the extra processing delay due to the introduction of the middle layer. FlexPM exhibits good performance compared with the other six schemes.

(4) Overall performance of all flows: Fig. 8(g) shows the overall performance of all flows. FlexPM performs close to Optimal and outperforms the other five solutions, since it enjoys more available control resource by sacrificing the path programmability of the normal flows that have limited impact on the overall performance of all flows. The comparison between FlexPM and PM also proves that the newly proposed solution to configure proper normal flows under legacy mode plays an important role in offline flows recovery.

Computation time: We further evaluate the computation time of FlexPM and Optimal under the above two scenarios, and Table VI shows the result. In the figure, the computation time of FlexPM is only 19.18% and 18.90% of Optimal on average under two failure scenarios. Based on the outstanding performance of FlexPM in Figs. 7 and 8, the proposed FlexPM is proved to be an efficient heuristic solution with low computation complexity.

TABLE VI

PERCENTAGE OF COMPUTATION TIME OF FLEXPM TO OPTIMAL UNDER BELNET TOPOLOGY. THE LOWER, THE BETTER

Scenario	One controller failure (%)	Two controller failures (%)
FlexPM	19.18	18.90
Optimal	100.00	100.00

VII. RELATED WORK

A. Resilient Network Control

1) Static Solutions: In order to realize resilient network control, static solutions optimally select proper locations and place the SDN controller with the consideration of mitigating the possibility of potential controller failure and link failure. They also focus on reducing communication overhead between controllers and switches. Vizarreta et al. [26] present two strategies to address the Reliable Controller Placement (RCP) problem, which protects the control plane against link and node failures. Tivig et al. [27] attempt a critical study of different placement solutions to provide resiliency. Mohammed et al. [28] present a new metric to measure node's importance to determine the locations of controllers against network failure. Santos et al. [29] propose to find the feasible controller placement strategy to satisfy the QoS requirements and robustness against potential failures. PrePass-Flow [30] is Machine Learning (ML)-based scheme to predict link failures before their occurrence and take further steps to minimize the impact of network-layer failure. Hu et al. [31] analyze the real link data to get the characteristics of link failures and propose robust controller placement scheme to cope with link failures.

2) Dynamic Solutions: As for the dynamic solutions, they take the current status of controllers and switches in real time. They dynamically assign the switches to the controllers. He et al. [6] propose a master and slave controller assignment model, which can prevent multiple controller failures under the constraints of the propagation latency. They further introduce a priority policy to decide the master controllers in each

failure case [8]. ProgrammabilityGuardian [11] 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. Matchmaker [7] can adaptively adjust the control cost of offline switches based on the limited control resource by changing the paths of flows to realize efficient offline switches remapping. He et al. [32] focus on handling load balancing against multiple controller failures in SDN, and design a priority set to further minimize the maximum utilization ratio among controllers in the worst-case of failure patterns. Guillen et al. [33] analyze the multi-controller failure problem in hybrid wired/wireless networks, and introduce a three-stage resilient mechanism (*i.e.*, controller disconnection avoidance, data communication protection, and disaster impact monitoring) to alleviate the impact of multiple controller failure.

B. Controller-Switch Mapping

Tanha et al. [4] take both the switch-controller and inter-controller latency requirements and the capacity of the controllers into consideration to select the resilient placement of controllers. Wang et al. [34] propose DCAP to consider dynamic controller assignment so as 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 [35]. Huang et al. [36] propose a novel scheme to jointly consider both static and dynamic switch-controller association and devolution. Bera et al. [37] introduce a fine-grained traffic-aware controller assignment approach to minimize controller response time with the help of FlowVisor. Yang et al. [38] propose a new metric called residual capacity to represent the available control resource of controller, and focus on maximizing the residual capacity of controllers to deal with the traffic burstiness in the network. Tohidi et al. [39] concentrate on 5G network, and take the size of the network and users' dynamic behavior into considerations in their proposed controller assignment heuristic algorithm. Yuan et al. [40] focus on controller assignment problem in software defined internet of vehicles, and adopt deep reinforcement learning to efficiently solve the problem. Indirect Multi-Mapping (IMM) [38] is proposed to achieve both high residual capacity and low synchronization cost to deal with potential traffic dynamics and burstiness in SDN by leveraging the FlowVisor.

C. Hybrid SDN

The deployment of hybrid SDN can be divided into two main categories [41]. The first is to upgrade legacy networking devices to SDN switches in a legacy network. The second category is to employ hybrid SDN/legacy routing mode enabled by high-end commercial SDN switches in a network, where our proposed FlexPM belongs. For the second category, Vissicchio et al. [17] deploy hybrid SDN/legacy technique to build coexisting control-planes and provide detailed configuration guidelines. Xu et al. use hybrid routing mode to realize the improvement on network performance [18], [42]. RetroFlow [10] intelligently configures a set of selected offline switches working under the legacy routing mode to relieve the active controllers from controlling the selected offline switches while maintaining the flow programmability of SDN.

VIII. Conclusion and Future Work

In this paper, we propose FlexPM, a switch-level programmability recovery solution to recover offline flows with balanced programmability and improve the total programmability of offline flows. Thanks to the hybrid SDN/legacy routing mode supported by high-end commercial SDN switches, FlexPM approximately realizes the performance of the flow-level recovery solution by smartly deciding the routing mode of each offline flow at recovered switches and establishing effective mappings between offline switches and active controllers for programmability recovery.

Our method is designed for backbone networks, where traffic usually consists of coarse-grained two-tuple source-destination flows. Thus, our solution aims to limit propagation delay in general. For fine-grained five-tuple flows, we can add some QoS-based constraints (*e.g.*, propagation delay, bandwidth requirement) to formulate the problem for fine-grained routing policy updates. In the future, we will consider these metrics in maintaining path programmability for fine-grained flows with QoS requirements.

References

[1] S. Dou, Z. Guo, and Y. Xia, "ProgrammabilityMedic: Predictable path programmability recovery under multiple controller failures in SD-WANs," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2021, pp. 461–471.

[2] C.-Y. Hong et al., "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM*, 2013, pp. 15–26.

[3] S. Jain et al., "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM Conf.*, Aug. 2013, pp. 3–14.

[4] M. Tanha, D. Sajjadi, R. Ruby, and J. Pan, "Capacity-aware and delay-guaranteed resilient controller placement for software-defined WANs," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 3, pp. 991–1005, Sep. 2018.

[5] N. Perrot and T. Reynaud, "Optimal placement of controllers in a resilient SDN architecture," in *Proc. 12th Int. Conf. Design Reliable Commun. Netw. (DRCN)*, Mar. 2016, pp. 145–151.

[6] F. He, T. Sato, and E. Oki, "Master and slave controller assignment model against multiple failures in software defined network," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.

[7] S. Dou, G. Miao, Z. Guo, C. Yao, W. Wu, and Y. Xia, "Matchmaker: Maintaining network programmability for software-defined WANs under multiple controller failures," *Comput. Netw.*, vol. 192, Jun. 2021, Art. no. 108045.

[8] F. He and E. Oki, "Main and secondary controller assignment with optimal priority policy against multiple failures," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 4, pp. 4391–4405, Dec. 2021.

[9] G. Yao, J. Bi, and L. Guo, "On the cascading failures of multi-controllers in software defined networks," in *Proc. 21st IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2013, pp. 1–2.

[10] Z. Guo, W. Feng, S. Liu, W. Jiang, Y. Xu, and Z.-L. Zhang, "RetroFlow: Maintaining control resiliency and flow programmability for software-defined WANs," in *Proc. Int. Symp. Qual. Service*, Jun. 2019, pp. 1–10.

[11] Z. Guo, S. Dou, and W. Jiang, "Improving the path programmability for software-defined WANs under multiple controller failures," in *Proc. IEEE/ACM 28th Int. Symp. Qual. Service (IWQoS)*, Jun. 2020, pp. 1–10.

[12] R. Sherwood et al., "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium*, vol. 1, p. 132, Oct. 2009.

[13] *OpenFlow Switch Specification 1.3*. Accessed: Dec. 5, 2022. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[14] B. P. R. Killi and S. V. Rao, "Capacitated next controller placement in software defined networks," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 3, pp. 514–527, Sep. 2017.

[15] *Brocade MLX-8 PE*. Accessed: Dec. 5, 2022. [Online]. Available: https://www.dataswitchworks.com/datasheets/MLX_Series_DS.pdf

[16] *CHN-IX*. Accessed: Dec. 5, 2022. [Online]. Available: http://www.chn-ix.net/

[17] S. Vissicchio, L. Cittadini, O. Bonaventure, G. G. Xie, and L. Vanbever, "On the co-existence of distributed and centralized routing control-planes," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 469–477.

[18] H. Xu, H. Huang, S. Chen, and G. Zhao, "Scalable software-defined networking through hybrid switching," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[19] J. Xie, D. Guo, X. Li, Y. Shen, and X. Jiang, "Cutting long-tail latency of routing response in software defined networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 384–396, Mar. 2018.

[20] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[21] P. Thorat, R. Challa, S. M. Raza, D. S. Kim, and H. Choo, "Proactive failure recovery scheme for data traffic in software defined networks," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 219–225.

[22] M. Shojaee, M. Neves, and I. Haque, "SafeGuard: Congestion and memory-aware failure recovery in SD-WAN," in *Proc. 16th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2020, pp. 1–7.

[23] C. C. Robusto, "The cosine-haversine formula," *Amer. Math. Monthly*, vol. 64, no. 1, pp. 38–40, 1957.

[24] *Speed, Rates, Times, Delays: Data Link Parameters for CSE 461*. Accessed: Dec. 5, 2022. [Online]. Available: https://courses.cs.washington.edu/courses/cse461/99wi/issues/definitions.html

[25] *Gurobi Optimization*. Accessed: Dec. 5, 2022. [Online]. Available: http://www.gurobi.com

[26] P. Vizarreta, C. M. Machuca, and W. Kellerer, "Controller placement strategies for a resilient SDN control plane," in *Proc. IEEE RNDM*, 2016, pp. 253–259.

[27] P.-T. Tivig and E. Borcoci, "Critical analysis of multi-controller placement problem in large SDN networks," in *Proc. 13th Int. Conf. Commun. (COMM)*, Jun. 2020, pp. 489–494.

[28] M. J. F. Alenazi and E. K. Çetinkaya, "Resilient placement of SDN controllers exploiting disjoint paths," *Trans. Emerg. Telecommun. Technol.*, vol. 31, no. 2, p. e3725, Feb. 2020.

[29] D. Santos, T. Gomes, and D. Tipper, "SDN controller placement with availability upgrade under delay and geodiversity constraints," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 301–314, Mar. 2021.

[30] M. Ibrar, L. Wang, G.-M. Muntean, A. Akbar, N. Shah, and K. R. Malik, "PrePass-Flow: A machine learning based technique to minimize ACL policy violation due to links failure in hybrid SDN," *Comput. Netw.*, vol. 184, Jan. 2021, Art. no. 107706.

[31] T. Hu et al., "An efficient approach to robust controller placement for link failures in software-defined networks," *Future Gener. Comput. Syst.*, vol. 124, pp. 187–205, Nov. 2021.

[32] F. He and E. Oki, "Load balancing model against multiple controller failures in software defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–6.

[33] L. Guillen, S. Izumi, T. Abe, and T. Suganuma, "A resilient mechanism for multi-controller failure in hybrid SDN-based networks," in *Proc. 22nd Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Sep. 2021, pp. 285–290.

[34] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.

[35] T. Wang, F. Liu, and H. Xu, "An Efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2788–2801, Oct. 2017.

[36] X. Huang, S. Bian, Z. Shao, and H. Xu, "Dynamic switch-controller association and control devolution for SDN systems," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.

[37] S. Bera, S. Misra, and N. Saha, "Traffic-aware dynamic controller assignment in SDN," *IEEE Trans. Commun.*, vol. 68, no. 7, pp. 4375–4382, Jul. 2020.

[38] X. Yang et al., "Indirect multi-mapping for burstiness management in software defined networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2059–2072, May 2021.

[39] E. Tohidi, S. Parsaeefard, A. A. Hemmati, M. A. Maddah-Ali, B. H. Khalaj, and A. Leon-Garcia, "Distributed controller-switch assignment in 5G networks," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 2193–2207, Mar. 2021.

[40] T. Yuan, W. D. R. Neto, C. E. Rothenberg, K. Obraczka, C. Barakat, and T. Turletti, "Dynamic controller assignment in software defined internet of vehicles through multi-agent deep reinforcement learning," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 585–596, Mar. 2021.

[41] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN networks: A survey of existing approaches," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3259–3306, 4th Quart., 2018.

[42] H. Xu, H. Huang, S. Chen, G. Zhao, and L. Huang, "Achieving high scalability through hybrid switching in software-defined networking," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 618–632, Feb. 2018.

Zehua Guo (Senior Member, IEEE) received the B.S. degree from Northwestern Polytechnical University, Xi'an, China, the M.S. degree from Xidian University, Xi'an, and the Ph.D. degree from Northwestern Polytechnical University. He was a Research Fellow at the Department of Electrical and Computer Engineering, Tandon School of Engineering, New York University, 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 and network function virtualization), machine learning, and network security. He 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., *Computer Communications* (Elsevier), AAAI, IWQoS, ICC, ICCCN, and ICA3PP). He is a Senior Member of China Computer Federation, China Institute of Communications, Chinese Institute of Electronics, and a member of ACM.

Songshi Dou (Graduate Student Member, IEEE) received the B.S. degree from North China Electric Power University, Beijing, China, in 2019, and the M.S. degree from the Beijing Institute of Technology, Beijing, in 2022. His research interests include software-defined networking, network function virtualization, and data center networks. He is a Graduate Student Member of ACM.

Wenfei Wu (Member, IEEE) received the B.S. degree from Beihang University, Beijing, China, in 2010, and the Ph.D. degree from the Computer Sciences Department, University of Wisconsin-Madison, Madison, Wisconsin, in 2015. He is currently an Assistant Professor at the School of Computer Science, Peking University. His research interests include networked systems.

Yuanqing Xia (Senior Member, IEEE) 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 Post-Doctoral 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, as an Associate Professor. Since 2008, he has also been a Professor. His current research interests include the fields of networked control systems, robust control and signal processing, and active disturbance rejection control.