# Improving the Path Programmability for Software-Defined WANs under Multiple Controller Failures

*IEEE/ACM International Symposium on Quality of Service*

Zehua Guo[1], Songshi Dou[1] & Wenchao Jiang [2]

[1] Beijing Institute of Technology, Beijing, China
[2] Singapore University of Technology and Design, Singapore


Presented by:
Songshi Dou
16th June 2020

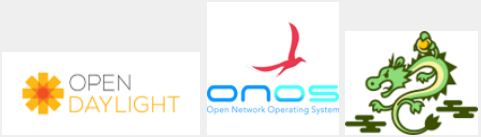
https://iwqos2020.ieee-iwqos.org/

# Presentation Outline

- Date plane design
  1. Programmable switch
  2. White-box switch

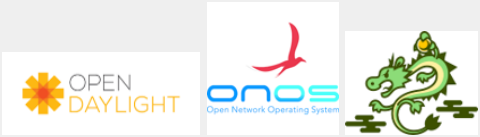
- Date plane design
  1. Programmable switch
  2. White-box switch


- Control plane design
  1. Single controller
  2. Controller cluster



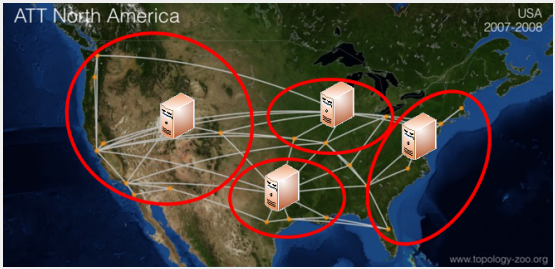
- Date plane design
  1. Programmable switch
  2. White-box switch


- Control plane design
  1. Single controller
  2. Controller cluster

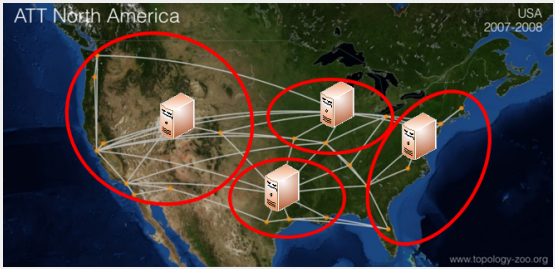



- Real deployment

- Large scale with many devices

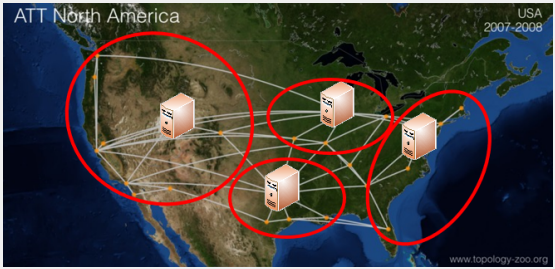


- Large scale with many devices
- Partitioning the network into domains



# Software-Defined Wide Area Networks (SD-WANs)

- Large scale with many devices
- Partitioning the network into domains
- Distributed control plane
  Quick response
  Control resiliency

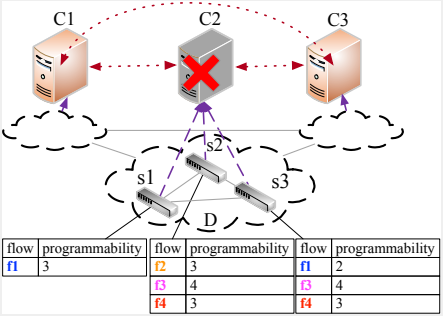


# Presentation Outline

## Maintaining Control Resiliency for Multiple Switches

### Controller failures

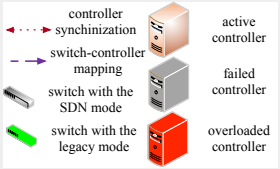
- Software bugs, Hardware failure, Power outage




| flow | programmability |
|---|---|
| f1 | 3 |

| flow | programmability |
|---|---|
| f2 | 3 |
| f3 | 4 |
| f4 | 3 |

| flow | programmability |
|---|---|
| f1 | 2 |
| f3 | 4 |
| f4 | 3 |



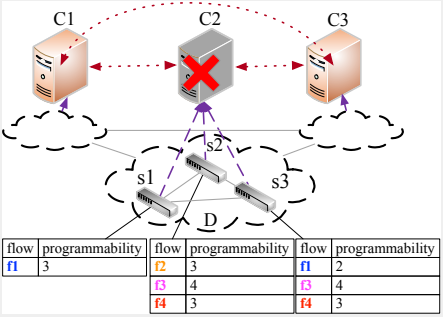
## Maintaining Control Resiliency for Multiple Switches

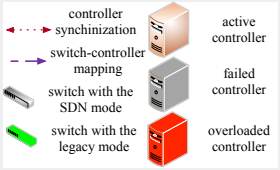
### Controller failures

- Software bugs, Hardware failure, Power outage

### Maintaining control resiliency

- Backup/slave controller
- Failed switches remapping



## Maintaining Control Resiliency for Multiple Switches

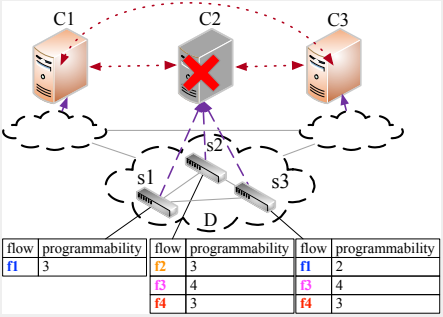
### Controller failures

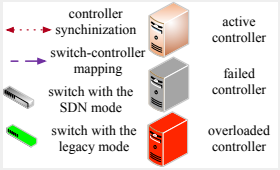
- Software bugs, Hardware failure, Power outage

### Maintaining control resiliency

- Backup/slave controller
- Failed switches remapping

### Programmability



# Presentation Outline

## Static Solutions

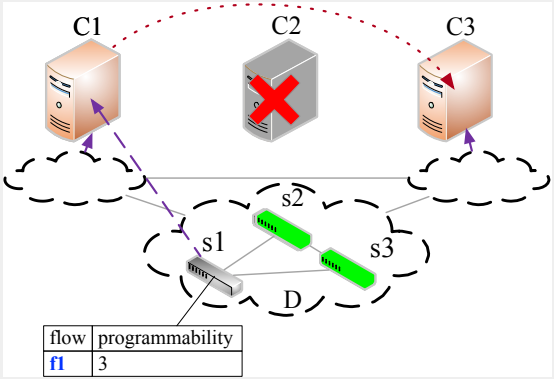
Statically placing, selecting, and mapping backup controllers to switches before controller failures.

## Static Solutions

Statically placing, selecting, and mapping backup controllers to switches before controller failures.

## Dynamic Solutions

**RetroFlow** dynamically sets up some offline switches work under the legacy routing mode without the controllers and maps the rest offline switches with the SDN routing mode to active controllers.



# Presentation Outline

**_Unbalanced programmability_ of offline flows**

With the switch-level mapping, some recovered flows can have high programmability of multiple rerouting paths while others are not recovered and cannot be rerouted at all.

### *Unbalanced programmability* of offline flows

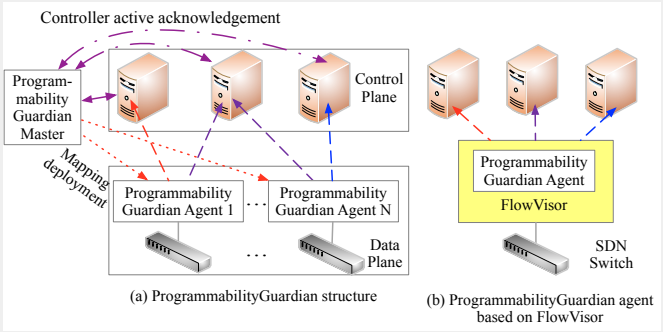
With the switch-level mapping, some recovered flows can have high programmability of multiple rerouting paths while others are not recovered and cannot be rerouted at all.

### *Under utilization* of active controllers

The switchlevel mapping solutions may cause the controller underutilization, which fails to map some offline flows to the active controllers even when the active controllers are not fully occupied.

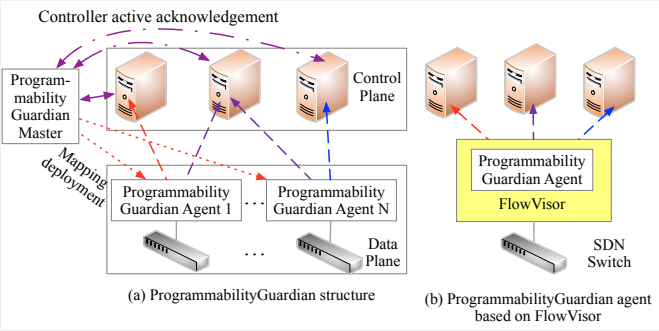
# Presentation Outline

PG aims at improving the path programmability in offline flow recovery under controller failures by realizing the **fine-grained flows to controllers mappings**.



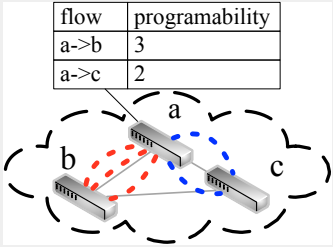
PG aims at improving the path programmability in offline flow recovery under controller failures by realizing the **fine-grained flows to controllers mappings**.

If one or multiple controllers are identified as failure, the master calculates *mappings* between offline flows from offline switches and active controllers and deploy the *mappings* into PG agents of offline switches.



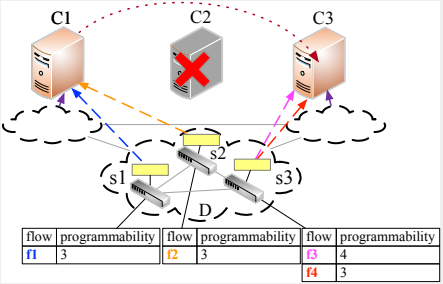
## Programmability calculating

The figure illustrates the **path programmability** of two flows on switch $a$. For switch $a$, the path programmability of one flow denotes the **ability of switch $a$ to change this flow's path**. For flow from $a$ to $b$, there are three paths traversing switch $a$, and its programmability is three on switch $a$. Similarly, flow from $a$ to $c$ has two paths on switch $a$, and its programmability is two on switch $a$.



## Constraints

- Controller processing ability for flow state pulling
- Flow programmability requirement




| flow | programmability | flow | programmability | flow | programmability |
|---|---|---|---|---|---|
| f1 | 3 | f2 | 3 | f3 | 4 |
| | | | | f4 | 3 |

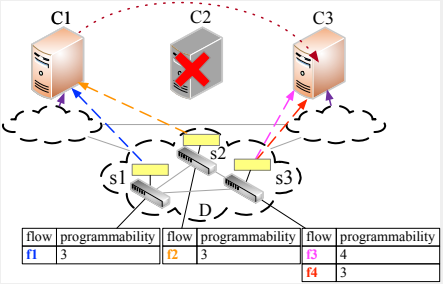
# Optimal Flow-Controller Mapping (OFCM) problem

## Constraints

- Controller processing ability for flow state pulling
- Flow programmability requirement

## Objective

- Maximize the recovered flows number and let each flow have the similar programmability
- Fully utilize the active controllers' control resource
- Minimize the conmunication overhead



# Optimal Flow-Controller Mapping (OFCM) problem

## Constraints

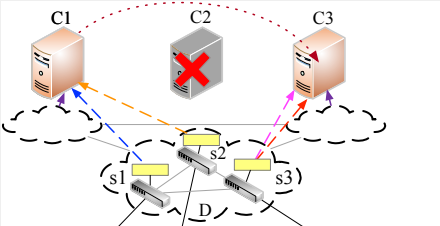
- Controller processing ability for flow state pulling
- Flow programmability requirement

## Objective

- Maximize the recovered flows number and let each flow have the similar programmability
- Fully utilize the active controllers' control resource
- Minimize the conmunication overhead

## Complexity

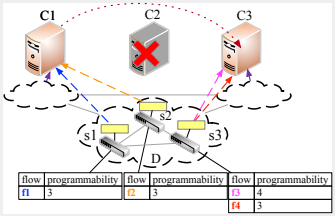
- An integer programming problem
- NP-hard




| flow | programmability | flow | programmability | flow | programmability |
|---|---|---|---|---|---|
| f1 | 3 | f2 | 3 | f3 | 4 |
| | | | | f4 | 3 |

## For all offline flows

- Flow $f$: Sorting the result of the Linear Programming relaxation of OFCM problem in the descending order of flow $f$.
- Controller $C$: Decreasing the control overhead of flow $f$.
- Flow-controller pair: $(f, C)$

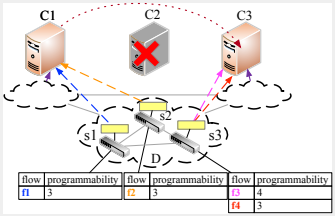


## For all offline flows

- Flow $f$: Sorting the result of the Linear Programming relaxation of OFCM problem in the descending order of flow $f$.
- Controller $C$: Decreasing the control overhead of flow $f$.
- Flow-controller pair: $(f, C)$

## Flow selection

- Testing flows based on the ascending order of their programmability in order to let each flow have the similar programmability.



## For all offline flows

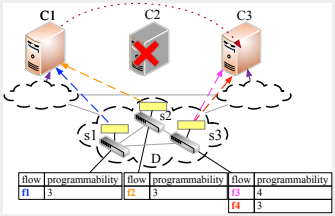
- Flow $f$: Sorting the result of the Linear Programming relaxation of OFCM problem in the descending order of flow $f$.
- Controller $C$: Decreasing the control overhead of flow $f$.
- Flow-controller pair: ($f$, $C$)

## Flow selection

- Testing flows based on the ascending order of their programmability in order to let each flow have the similar programmability.

## Controller assignment

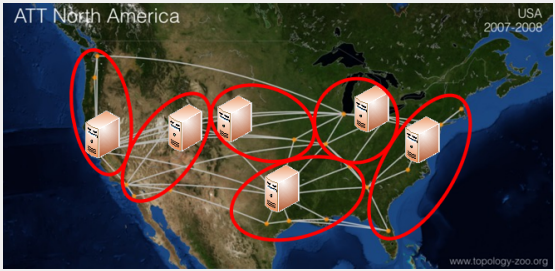
- Assigning the flow to the controller, which is based on order above and keeps enough processing ability.



# Presentation Outline

## Simulation setup

- AT&T topology with 25 nodes and 112 (56*2) links
- 6 controllers
- Any two nodes have a flow

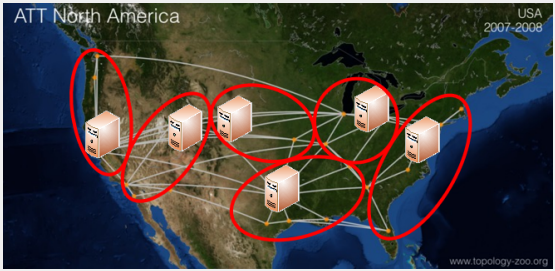


## Simulation setup

- AT&T topology with 25 nodes and 112 (56*2) links
- 6 controllers
- Any two nodes have a flow

## Comparison algorithms

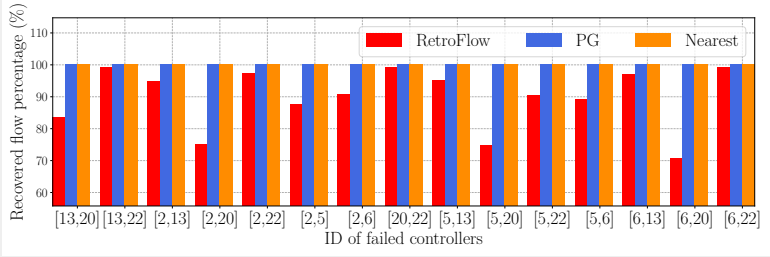
- Nearest
- RetroFlow
- ProgrammabilityGurdian



Two controllers failure

## Performance metric

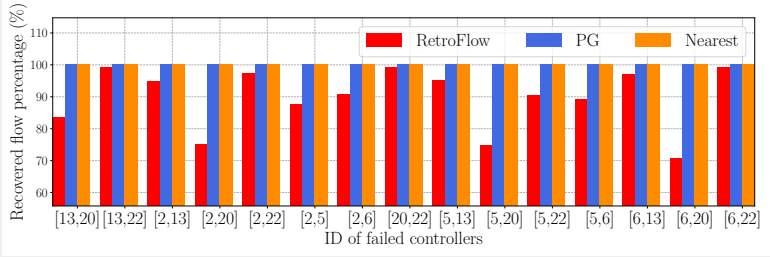
- Recovered flow percentage from offline flows.



Two controllers failure

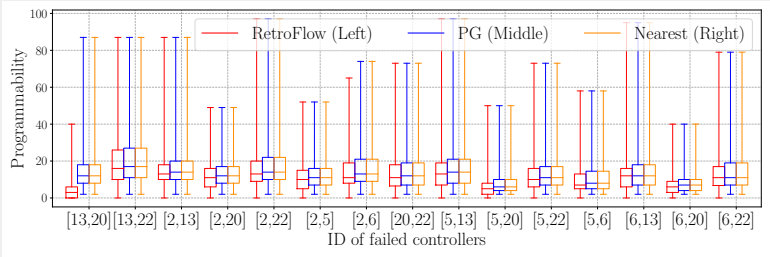
## Performance metric

- Recovered flow percentage from offline flows.




## Programmability metric

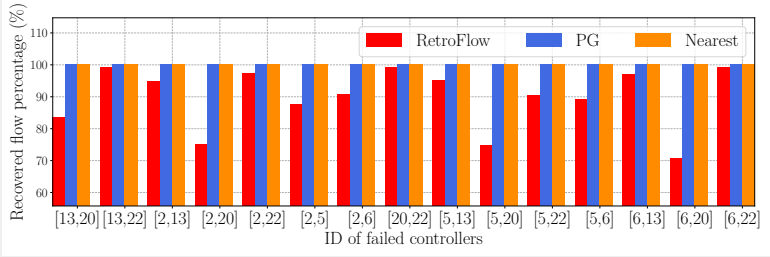
- The flow's path programmability.



Two controllers failure

## Performance metric

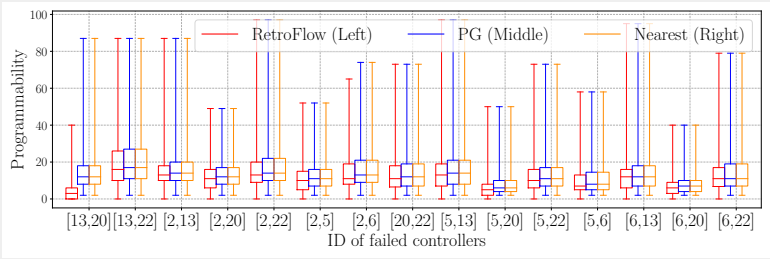
- Recovered flow percentage from offline flows.




## Programmability metric

- The flow's path programmability.

- More results in the paper.



# Presentation Outline

## New idea

- We propose **ProgrammabilityGuardian** to improve the *path programmability* of recovered flows with *low communication overhead* under multiple controller failures through the **fine-grained flow-level remapping** enabled in existing SDN techniques.

# Summary

## New idea

- We propose **ProgrammabilityGuardian** to improve the *path programmability* of recovered flows with *low communication overhead* under multiple controller failures through the **fine-grained flow-level remapping** enabled in existing SDN techniques.

## New problem and solution

- We formulate the flow recovery problem as an optimization problem called **OFCM problem** and propose an **efficient heuristic algorithm** to solve the problem.

# Summary

## New idea

- We propose **ProgrammabilityGuardian** to improve the *path programmability* of recovered flows with *low communication overhead* under multiple controller failures through the **fine-grained flow-level remapping** enabled in existing SDN techniques.

## New problem and solution

- We formulate the flow recovery problem as an optimization problem called **OFCM problem** and propose an **efficient heuristic algorithm** to solve the problem.

## Good performance

- We evaluate the performance of PG under *different controller failure scenarios*. Simulation results show that PG recovers **all offline flows** with a **balanced path programmability**.

Thank You!
Questions?