Inter-Flow Consistency: Novel SDN Update Abstraction for Supporting Inter-Flow Constraints

Weijie Liu* (wliu43@illinois.edu),
Rakesh B. Bobba†, Sibin Mohan*, Roy H. Campbell*

*University of Illinois at Urbana-Champaign
†Oregon State University
Outline

• SDN & Inter-flow Consistency

• Our Approach

• Experiments

• Conclusion
Software Defined Networks (SDN)

- **Decouple** Control Plane and Data Plane
- Controller installs **forwarding rules** in switches
Update Forwarding Rules

Old Rules

Update

New Rules

: old rule
: new rule
Update Inconsistency

- **Fail** to update all the devices **at the same time**
- Packets processed by both **old & new** rules
- **Problems:**
  - loops, black hole, congestion ...

```
• old rule
• new rule
```
Update Inconsistency

• **Fail** to update all the devices **at the same time**
• Packets processed by both **old** & **new** rules
• **Problems:**
  – loops, black hole, congestion ...
• **Existing Solutions***:
  – **Per-packet Consistency**: processed by either old or new
  – **Per-flow Consistency**: processed by either old or new

Inter-flow Constraints

• Enforcing constraints across different flows
  – for Security or Reliability
  – Ex 1. Power Grid: isolation of critical control flows from engineering flows
  – Ex 2. Network Operator: isolation between data flows of different companies
  – Ex 3. Data Center: related data flows need to be updated at the same time

**Question:** Will these constraints be respected during SDN updates?
Example I

- Security Policy: f1 & f2 should **NOT** pass through the same link

(a) Original Configuration

(b) Target Configuration
Example 1

- What if \( f_2 \) gets updated before \( f_1 \)?

(a) Original Configuration  (b) Target Configuration

(c) Transitional Configuration
Example II* 

- **H1 & H2**: first inspected, then talk with each

![Diagram](image)

(a) Original Configuration  
(b) Target Configuration

Example II*

- What if f2 gets updated before f1?

f1 & f2 should be updated at the same time; not guaranteed by existing update mechanisms
Observation:

Inter-flow constraints may be violated during SDN updates.

Problem:

Can we schedule update operations to guarantee inter-flow constraints during updates?
Theoretical Abstraction

• We propose: **Interflow Consistency**

**Spatial Isolation:**

- Packets from different flows cannot pass through the same link or device

**Version Isolation:**

- Packets from different related flows cannot be processed by two different versions of flow rules
Our Approach: 3 steps

Step I: Construct **Dependency Graph** to model updates

Step II: **Revised** Dependency Graph for inter-flow consistency

Step III: **Output** valid update order
Step I: Construct Dependency Graph*

- **Dependency Graph** (DG)
  - 3 types of node:
    - Operation Node (add/delete/modify a rule)
    - Path Node (links passed by a flow)
    - Resource Node (link bandwidth)

- Direction of edge between 2 nodes:
  - Resource Consumption
  - Operation Dependency

Step I: Example I

(a) Original Configuration
(b) Target Configuration

We need 4 operations nodes:

<table>
<thead>
<tr>
<th>ID</th>
<th>Entity</th>
<th>Update Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$S_3$</td>
<td>Add: forward $f_1$ to $S_4$</td>
</tr>
<tr>
<td>b</td>
<td>$S_1$</td>
<td>Modify: forward $f_1$ to $S_3$</td>
</tr>
<tr>
<td>c</td>
<td>$S_1$</td>
<td>Modify: forward $f_2$ to $S_4$</td>
</tr>
<tr>
<td>d</td>
<td>$S_2$</td>
<td>Delete rules of $f_2$</td>
</tr>
</tbody>
</table>

Also, 4 path nodes:
- $p_1$: $f_1$'s old path;
- $p_2$: $f_2$'s old path;
- $p_3$: $f_1$'s new path;
- $p_4$: $f_2$'s new path.

5 resource nodes for each link
Step I: construct DG

(a) Original Configuration

(b) Target Configuration

<table>
<thead>
<tr>
<th>ID</th>
<th>Entity</th>
<th>Update Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$S_3$</td>
<td>Add: forward $f_1$ to $S_4$</td>
</tr>
<tr>
<td>b</td>
<td>$S_1$</td>
<td>Modify: forward $f_1$ to $S_3$</td>
</tr>
<tr>
<td>c</td>
<td>$S_1$</td>
<td>Modify: forward $f_2$ to $S_4$</td>
</tr>
<tr>
<td>d</td>
<td>$S_2$</td>
<td>Delete rules of $f_2$</td>
</tr>
</tbody>
</table>
Step II: Revised DG for inter-flow

- e.g. Spatial Isolation: add Mutex Node
Step III: Output Operation Sequence

```plaintext
do:
  for each Operation Node, O:
    if O has no operation ancestors & has sufficient resources:
      Schedule O;
      Delete O;
    end if
  end for
until there is no O;
```

Finally, we can get: a→b→c→d
Packets from different related flows cannot be processed by two different versions of flow rules.

(a) Original Configuration

(b) Target Configuration

(c) Transitional Configuration
Solution for Version Isolation

- forward related packets to controller before updates

Algorithm in paper: implemented in DG

Experiments

• A prototype system
  – Spatial Isolation
  – Version Isolation being implemented

• Experiment settings:
  – Network Application: shortest-path routing
  – Control Plane: Ryu
  – Data Plane: Mininet, a 3-layer tree topology
  – Hardware: Intel i5-2400 3.1 GHz CPU & 16 GB memory
Experimental Results

(a) Number of Isolation Constraints & Update Operations on different Host Numbers

(b) Algorithm Running Time on different Host Numbers

Initial experiments show very good performance
Future Work

• Implementation of version isolation
  – optimized algorithm

• Evaluation
  – in real networks
  – in a large-scale simulation

• Further discussion: inter-flow consistency
  – relationship of two isolations
  – drawbacks
Conclusion

• Inter-flow consistency abstraction:
  – Spatial Isolation
  – Version Isolation

• An approach using dependency graph

• A prototype system
  – a preliminary performance evaluation
Questions?

• feel free to contact: wliu43@illinois.edu

Thank you!

• Thanks to Prof. Carl Gunter for slide template!
Two Consistency Abstractions

• Per-packet Consistency:
  – Each packet will be processed by the old configuration or the new, but NOT the mixture of the two.

  pkt pkt pkt pkt pkt pkt

• Per-flow Consistency:
  – Each flow will be processed by the old configuration or the new, but NOT the mixture of the two.

  pkt pkt pkt pkt pkt pkt
Spatial Isolation

• certain flows are not allowed to share a link or a switch before, during and after an update for security and/or reliability reasons.

• E.g. critical flows should be isolated from engineering flows
• packets from different related flows cannot be processed by two different versions of flow rules during its passage through the network.

• E.g. A flow’s rules updated from $R_{A1}$ to $R_{A2}$; another flow’s updated from $R_{B1}$ to $R_{B2}$; the network can have $R_{A1}R_{B1}$ or $R_{A2}R_{B2}$, but not $R_{A1}R_{B2}$ or $R_{A2}R_{B1}$
Enforcing Spatial Isolation

- Randomly generate flows between 2 hosts in the tree-like network (old & new)

- Brute Force Search:
  - for any flow A and another flow B: if they are spatially isolated both in old and new configuration, but not during the transitions (i.e., A’s old path overlaps with B’s new path) then assign a spatial isolation constraint to A and B.
Controller-buffer Method for Version Isolation\cite{4}

- Basic idea: use controller as a transitional point
  1. Forward packets to the controller
  2. Then update rules in switches
  3. Re-inject buffered packets from controller to data plane

For $N$ flows of Version Isolation, we can first forward $N-1$ flow to controller, then update rules. After all updates completed, controller sends buffered packets back to network.
DG Solution for Version Isolation

• After constructing basic DG, add operations to represent:

  (1) forward certain flows to controller
  (2) controller sends buffered flows back to network

• Then perform the topological sorting of operations
DG Solution for Version Isolation

- After constructing basic DG, add operations:
  - e: forward packets to controller;
  - f: delete rules of “e”;
  - g: controller send packets back to network.

- We can get:
  \[ e \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow f \rightarrow g \]
Update Order Consideration

- S1:
  - Floodgate node, change the path of flow

We should get this part ready before update the “floodgate”
Selected References


