LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

MNM TEAM

# Conflict Detection in Software-Defined Networks

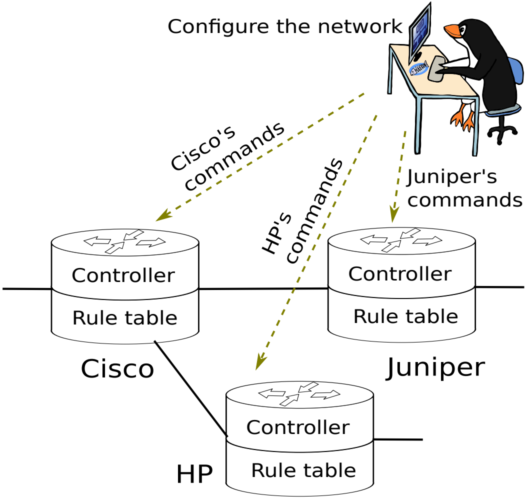

Cuong Ngoc Tran

mnm-team.org/∼cuongtran


Erster Gutachter: PD Dr. Vitalian Danciu
Zweiter Gutachter: Prof. Dr. Wolfgang Hommel
Vorsitzender: Prof. Dr. Matthias Schubert

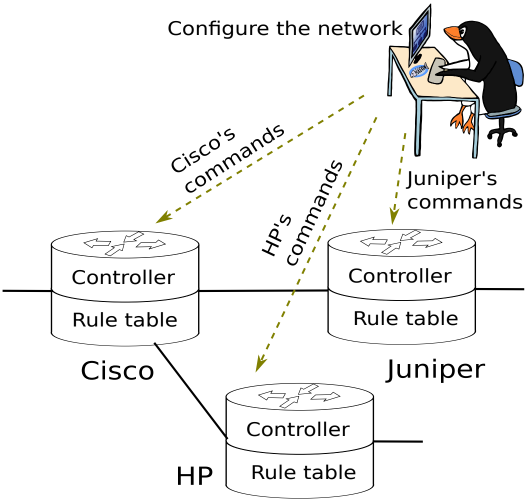
June 13, 2022

## Traditional networks

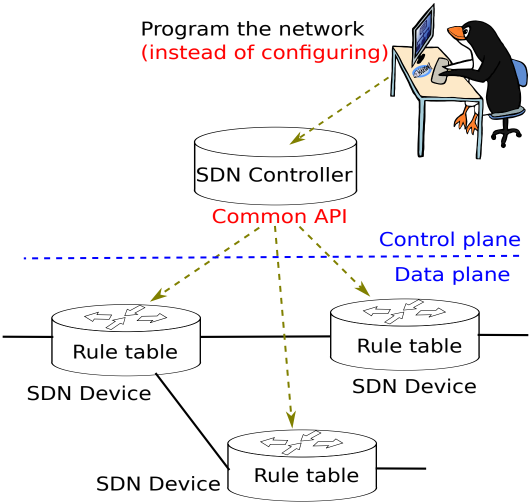


# Traditional Networks and SDN

## Traditional networks

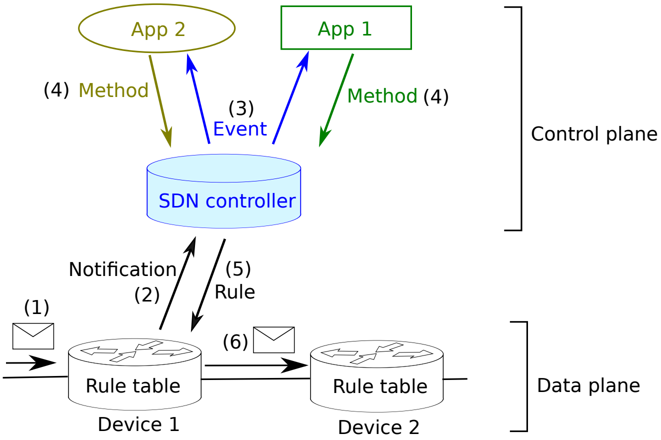



## Software-Defined Networks (SDN)



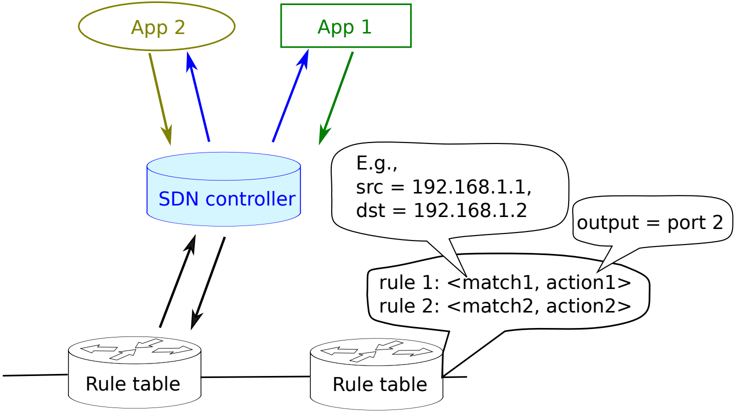
# SDN Architecture





# SDN Architecture





# Conflicts in Real Life



I want a grandchild

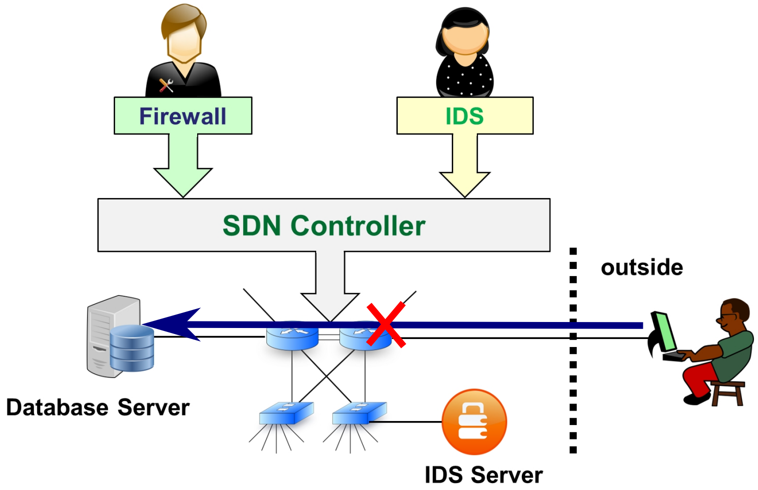
Deadline!

...

Take a walk, please

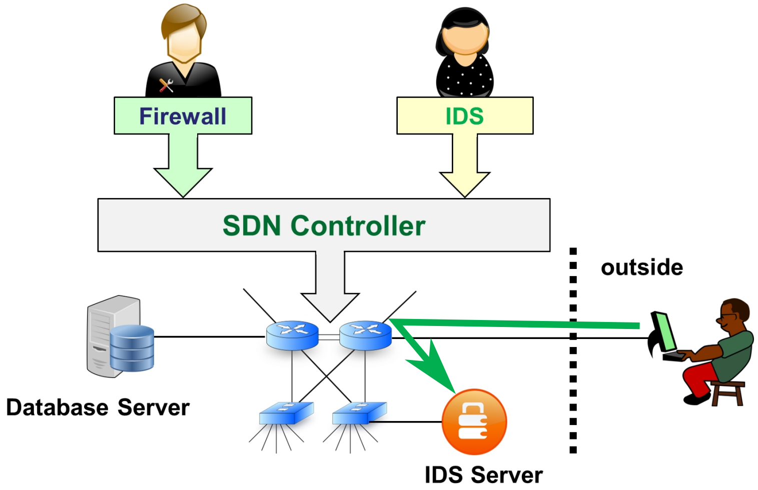
# Conflicts in SDN



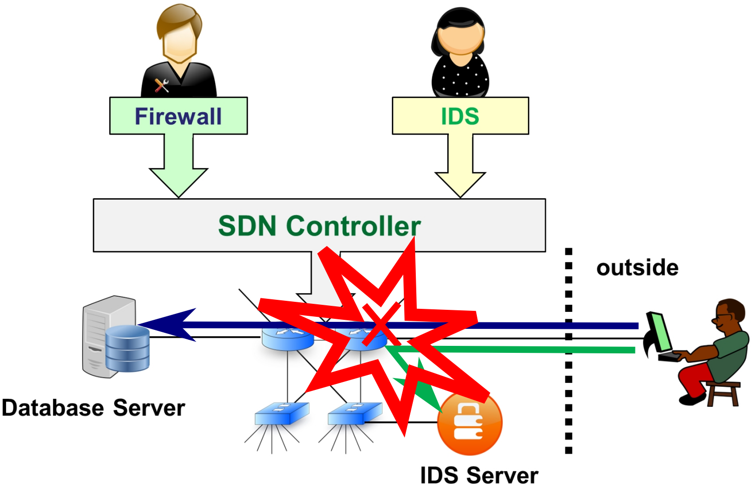


# Conflicts in SDN





# Conflicts in SDN



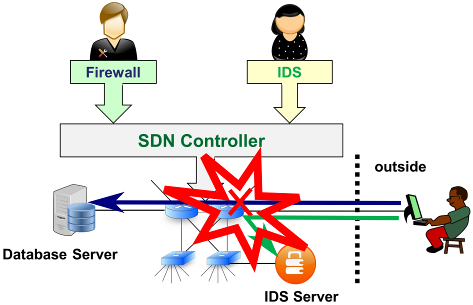
# Conflicts in SDN



**Possible consequences:**

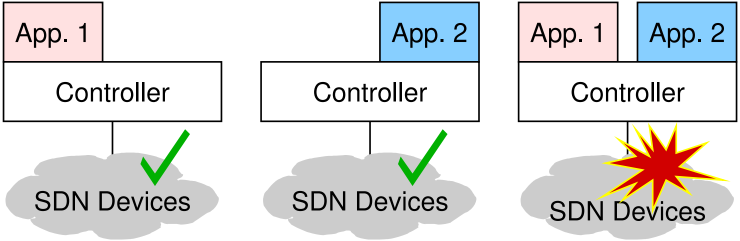
- Application's goals are not fulfilled
- Unexpected, unreliable network behaviour

⇒ Conflicts need to be detected and resolved



# Conflict Definition





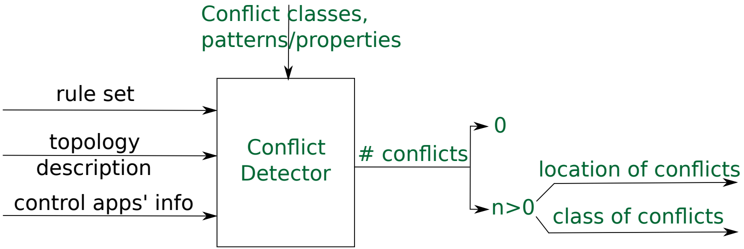
# Research Questions



1. What is a suitable method to research conflicts in SDN?
2. How can conflicts between control applications be classified based on their rules (conflict classification)?

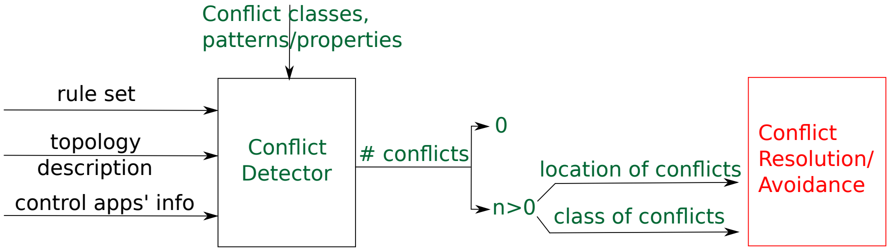
# Research Questions

1. What is a suitable method to research conflicts in SDN?
2. How can conflicts between control applications be classified based on their rules (conflict classification)?
3. How many conflicts exist in a given rule set (conflict detection)?
   3.1 Which rules cause conflicts?
   3.2 To which class does each detected conflict belong?



# Research Questions

1. What is a suitable method to research conflicts in SDN?
2. How can conflicts between control applications be classified based on their rules (conflict classification)?
3. How many conflicts exist in a given rule set (conflict detection)?
   3.1 Which rules cause conflicts?
   3.2 To which class does each detected conflict belong?

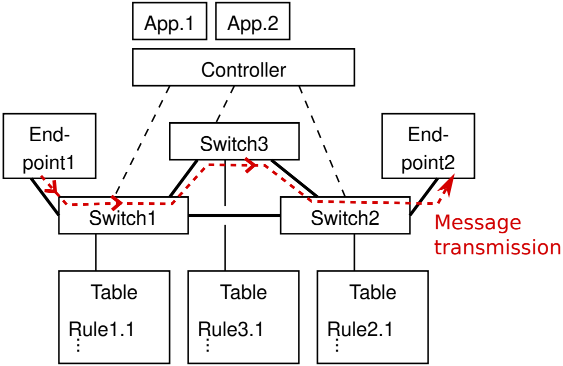


# Research Question 1

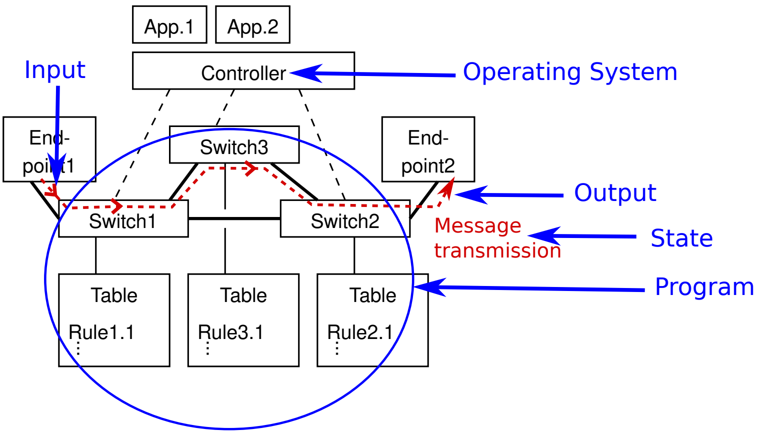


1. What is a suitable method to research conflicts in SDN?
2. How can conflicts between control applications be classified based on their rules (conflict classification)?
3. How many conflicts exist in a given rule set (conflict detection)?
   3.1 Which rules cause conflicts?
   3.2 To which class does each detected conflict belong?

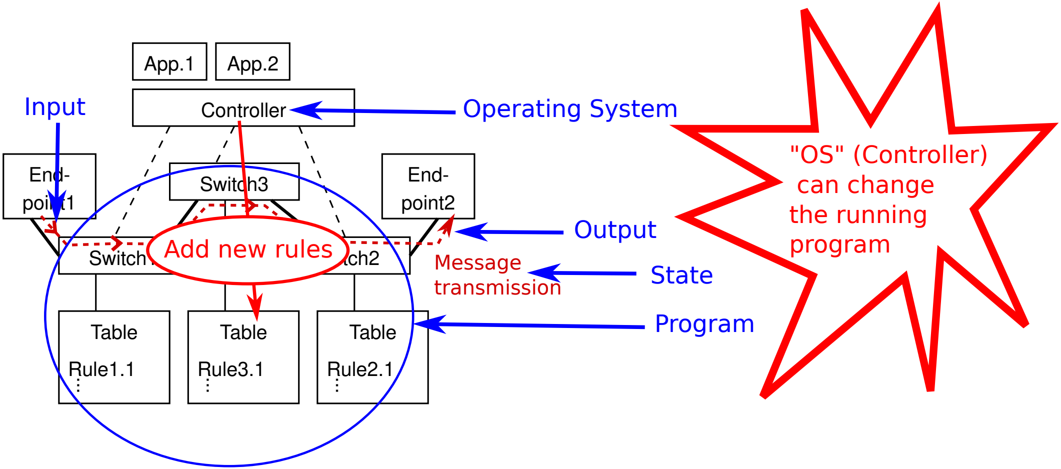
## Analytical approach based on a comparison with distributed computing systems



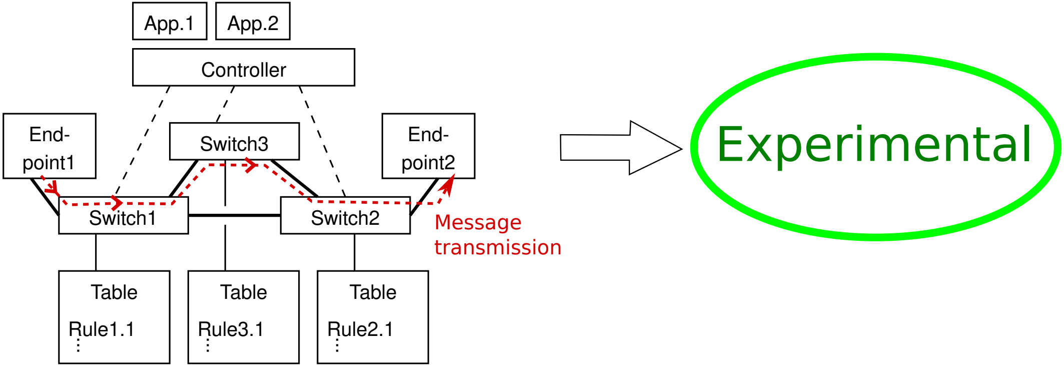
## Analytical approach based on a comparison with distributed computing systems



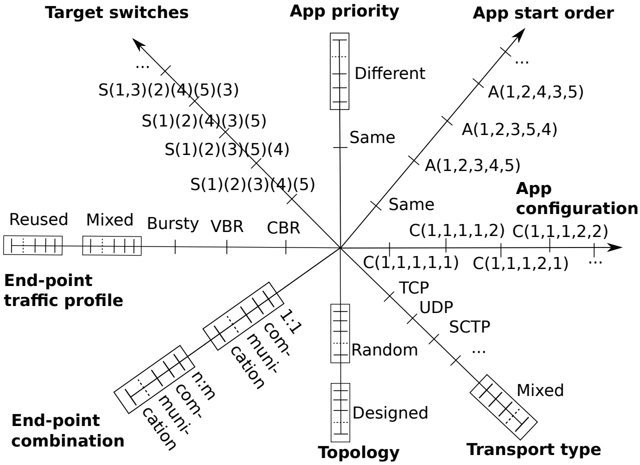
## Analytical approach based on a comparison with distributed computing systems



## Analytical approach based on a comparison with distributed computing systems



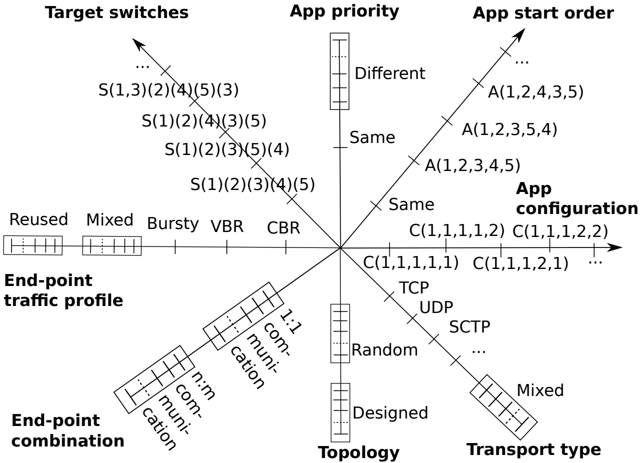
# Space for Experiments




Control applications:

- Shortest Path First Routing (SPF)
- End-point Load Balancer (EpLB)
- Path Load Balancer (PLB)
- Firewall (FW)
- . . .

# Space for Experiments

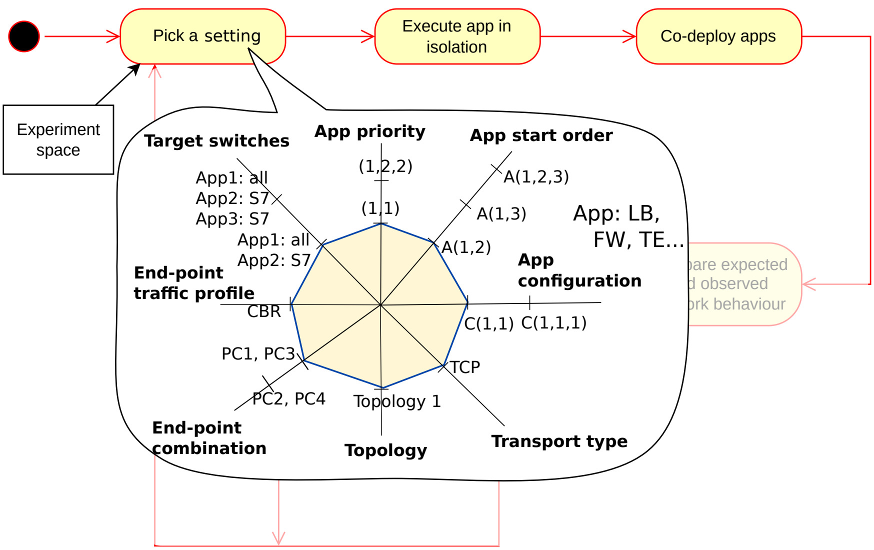



The number of experiments is immense

⇒ **restrict the space size and automate experiments**

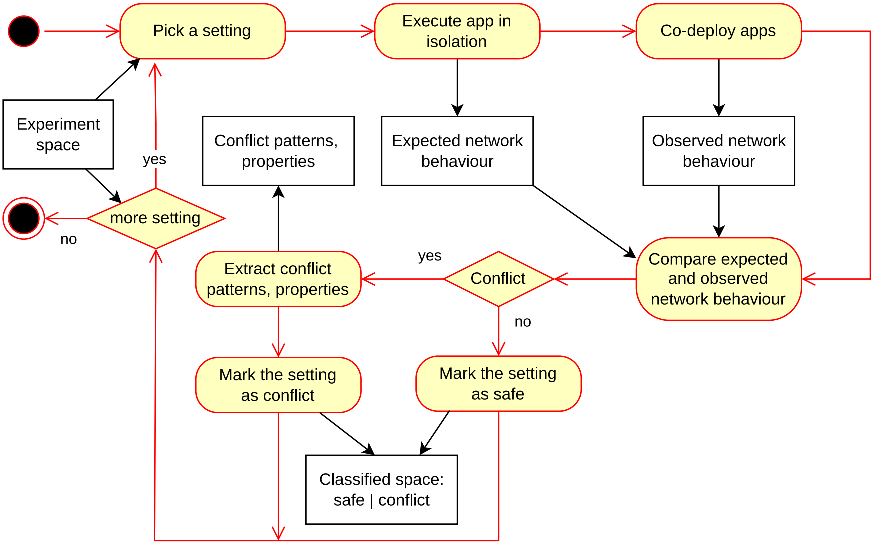
# Methodology for Experiments



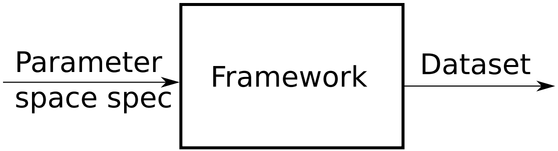


# Methodology for Experiments

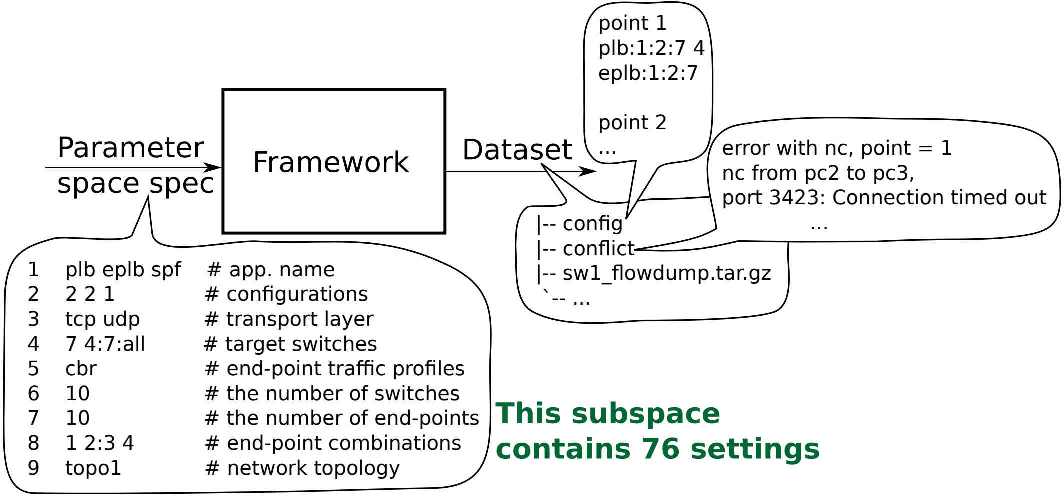




# Framework for automating experiments



# Framework for automating experiments

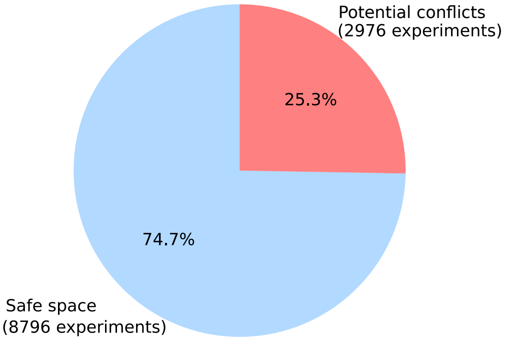



**This subspace contains 76 settings**

# Explored Subspaces



| | |
|---|---|
| # Topologies | 12 |
| # Applications | 14 |
| App. configuration | 1 → 5 |
| App. start order | same and different |
| App. priority | same and different |
| Target switches | 1 → all |
| Ep. Traffic Profile | CBR and VBR |
| EP. Combination | unicast, multicast |
| Transport type | TCP, UDP |
| # Experiments | 11,772 |

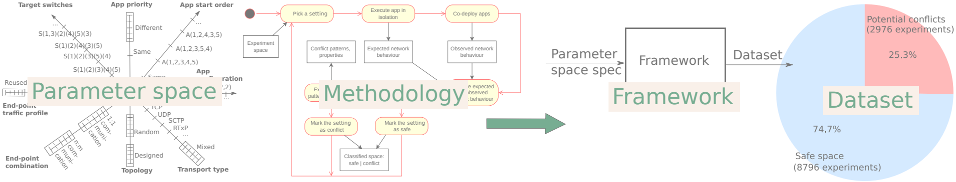



Dataset is available at
https://github.com/mnm-team/sdn-conflicts

# Research Question 2



1. What is a suitable method to research conflicts in SDN?




2. How can conflicts between control applications be classified based on their rules (conflict classification)?
3. How many conflicts exist in a given rule set (conflict detection)?
   3.1 Which rules cause conflicts?
   3.2 To which class does each detected conflict belong?

# Conflict Taxonomy

Conflicts in SDN

## Local Conflicts

- Shadowing
- Redundancy
- Correlation
- Overlap

**Caused by rules residing in a single device (local)**

## Distributed Conflicts

- Downstream Traffic Loop
- Downstream Traffic Drop
- Downstream Packet Modification
- Upstream Packet Modification
- Changes to Paths

**Caused by rules residing in different devices (distributed)**

## Hidden Conflicts

- Event Suppression by Local Handling
- Tampering with Event Subscription

**Causes cannot be discerned by rules in the devices (hidden), control plane's mechanics need to be examined in addition**

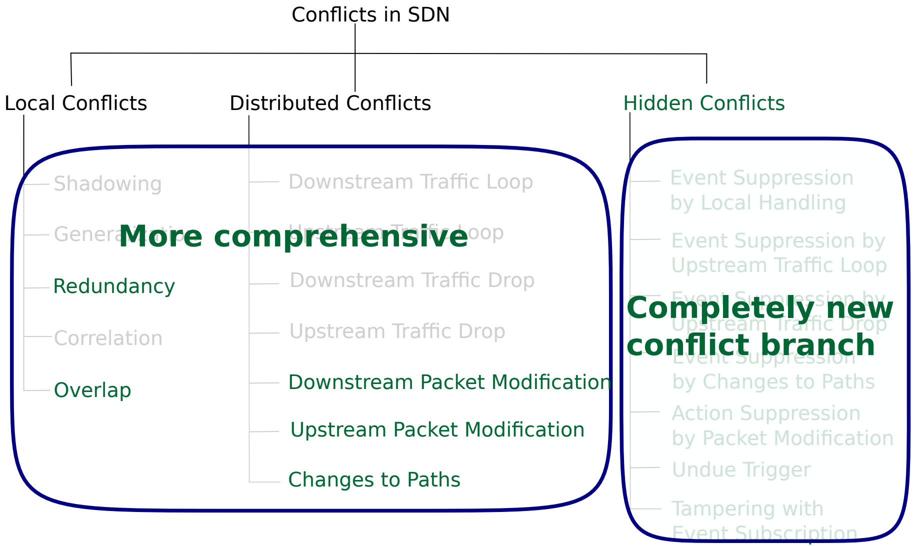
# Conflict Taxonomy

Conflicts in SDN

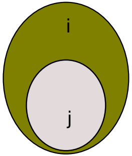
- Local Conflicts
  - Shadowing
  - Generalization
  - Redundancy
  - Correlation
  - Overlap
- Distributed Conflicts
  - Downstream Traffic Loop
  - Upstream Traffic Loop
  - Downstream Traffic Drop
  - Upstream Traffic Drop
  - Downstream Packet Modification
  - Upstream Packet Modification
  - Changes to Paths
- Hidden Conflicts
  - Event Suppression by Local Handling
  - Event Suppression by Upstream Traffic Loop
  - Event Suppression by Upstream Traffic Drop
  - Event Suppression by Changes to Paths
  - Action Suppression by Packet Modification
  - Undue Trigger
  - Tampering with Event Subscription

# Conflict Taxonomy



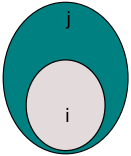


# Example: Local Conflicts



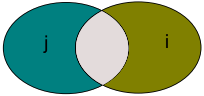

Action i ≠ Action j ⇒ Generalization

Action i = Action j ⇒ Overlap1



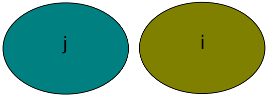

Action i = Action j ⇒ Redundancy

Action i ≠ Action j ⇒ Shadowing




Action i ≠ Action j ⇒ Correlation2

Action i = Action j ⇒ Overlap2




Any action ⇒ No conflict

Legend:

- Match space of rule i
- Match space of rule j
- Intersection of the match spaces of rules i and j

**priority i < priority j**

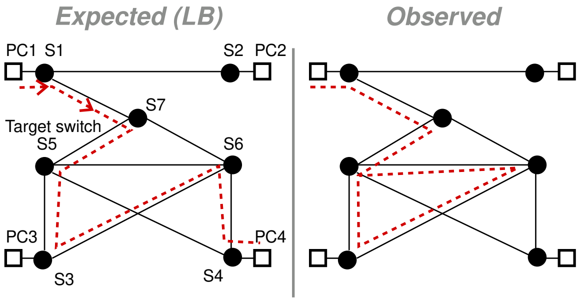
E.g., Rule i: pri = 2, src=192.168.1.**1**, dst=192.168.1.**2**, action=outport:1

Rule j: pri = 3, src=192.168.1.**1**, dst= **any**, action=outport:2

# Example: Distributed Conflicts



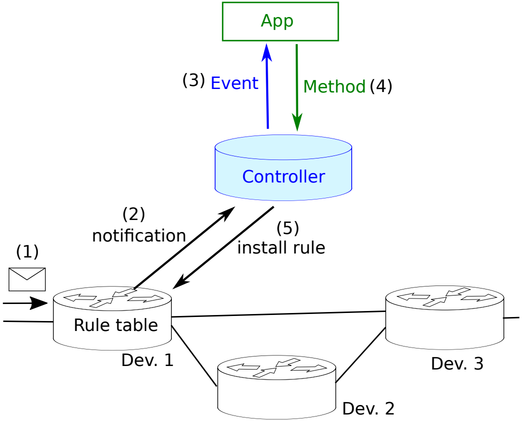
1. Downstream traffic loop
2. Upstream traffic loop
3. Downstream traffic drop
4. Upstream traffic drop
5. Downstream packet modification
6. Upstream packet modification
7. Changes to paths



# Hidden Conflicts' Classes



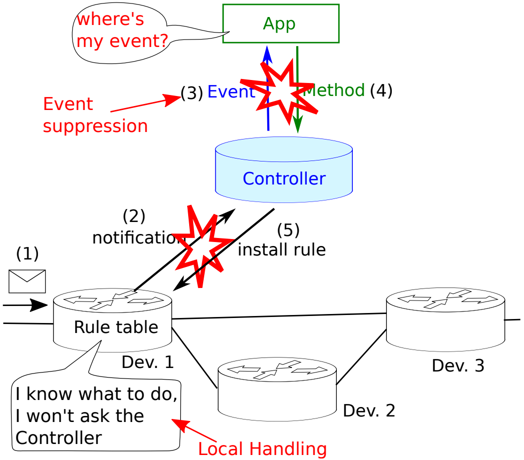
1. Event suppression by local handling
2. Event suppression by upstream traffic loop
3. Event suppression by upstream traffic drop
4. Event suppression by changes to paths
5. Action suppression by modification of packets
6. Undue trigger
7. Tampering with event subscription



# Hidden Conflicts' Classes

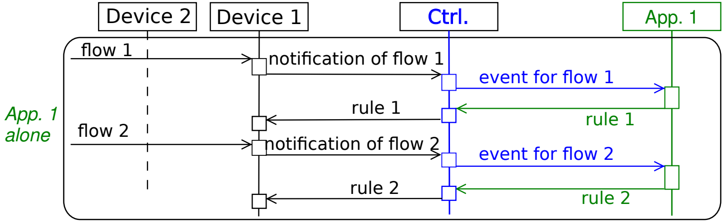


1. Event suppression by local handling
2. Event suppression by upstream traffic loop
3. Event suppression by upstream traffic drop
4. Event suppression by changes to paths
5. Action suppression by modification of packets
6. Undue trigger
7. Tampering with event subscription



# Example: Hidden Conflicts






rule : <match, action>

Example:

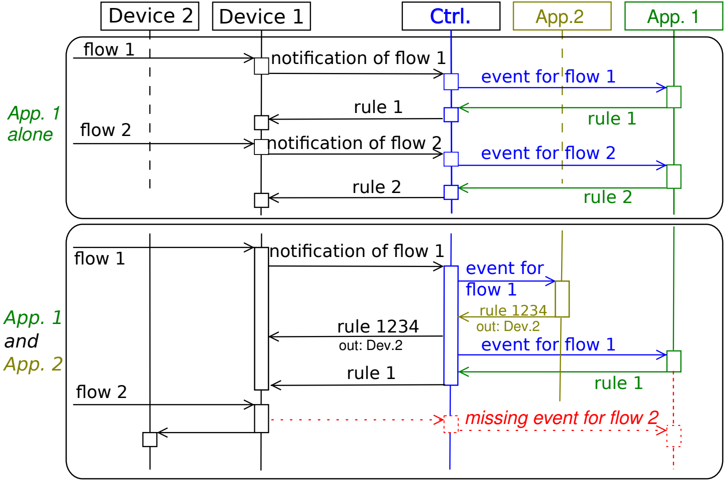
rule 1: <src = 192.168.1.**1**, dst = 192.168.1.**2**, action=port **1**>

rule 2: <src = 192.168.1.**1**, dst = 192.168.1.**3**, action=port **1**>

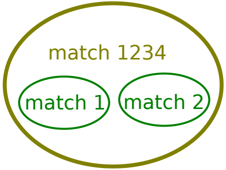
rule 1234: <src = 192.168.1.**1**, dst = **any**, action=port **2**>

# Example: Hidden Conflicts



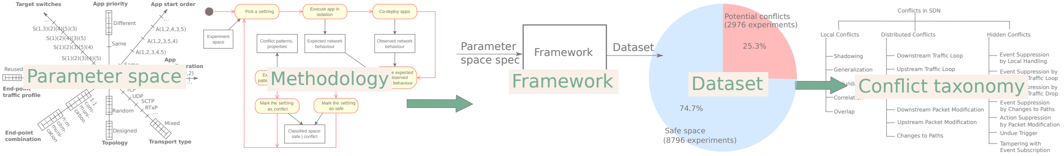



rule : <match, action>



# Research Question 3

1. What is a suitable method to research conflicts in SDN?
2. How can conflicts between control applications be classified based on their rules (conflict classification)?




3. How many conflicts exist in a given rule set (conflict detection)?
   3.1 Which rules cause conflicts?
   3.2 To which class does each detected conflict belong?

# Overview of the Measures



- Comparison of rules based on newly introduced concepts:
  - multi-property set
  - relationship combination operator “dot r” ($\cdot r$)
  - matchmap and actmap

# Overview of the Measures

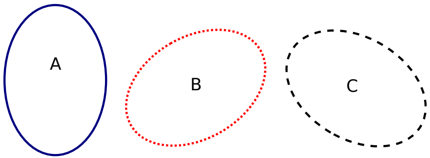


- Comparison of rules based on newly introduced concepts:
  - multi-property set
  - relationship combination operator “dot r” ($\cdot r$)
  - matchmap and actmap
- Rule graph
- Input from control applications
- Algorithms

# Multi-Property Set



$A$ = a set of flowers having **five petals**
$B$ = a set of flowers with **red color**
$C$ = a set of flowers being **scentless**

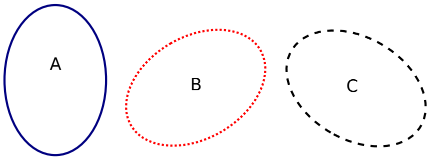



Question: $S_{ABC}$ = a set of flowers having **five petals, red color** and being **scentless** = ?

# Multi-Property Set

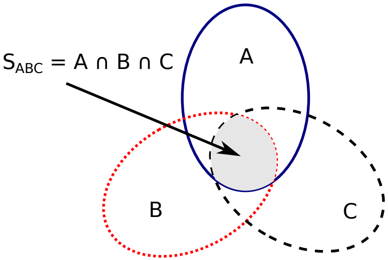


$A$ = a set of flowers having **five petals**
$B$ = a set of flowers with **red color**
$C$ = a set of flowers being **scentless**




Question: $S_{ABC}$ = a set of flowers having **five petals, red color** and being **scentless** = ?

Answer: $S_{ABC} = A \cap B \cap C$




**Match fields of SDN rules are multi-property sets**, e.g.,
match={ip_src=192.168.1.1, ip_dst=192.168.1.2,
ip_proto=tcp, tcp_dst=80}

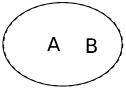
# Comparing Multi-Property Sets



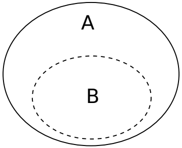
$$A = \{\text{color} \in \{\text{yellow}, \text{pink}, \text{red}, \text{blue}\}, \text{number of petals} > 5\} = A_{color} \cap A_{petal}$$

$$B = \{\text{color} \in \{\text{yellow}, \text{pink}\}, \text{number of petals} > 3\} = B_{color} \cap B_{petal}$$

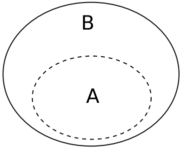
Question: what is the relationship of $A$ and $B$?



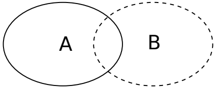

a) A and B are equal



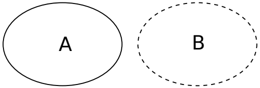

b) A is a proper superset of B




c) A is a proper subset of B




d) A intersects B




e) A and B are disjoint

## Relationship Combination Operator $\cdot r$



Relationship encoding: *disjoint* - 0, *equal* - 1, *proper subset* - 2, *proper superset* - 3, *intersecting* - 4

The operation of $\cdot r$:

$$\cdot r : (X, Y) \rightarrow Z, \ \ where \ X, Y, Z \in \{0, 1, 2, 3, 4\}$$
$$0 \ \cdot_r \ X = 0$$
$$X \ \cdot_r \ X = X$$
$$X \ \cdot_r \ 1 = X$$
$$2 \ \cdot_r \ 3 = 4$$
$$X \ \cdot_r \ 4 = 4 \ if \ X \neq 0$$

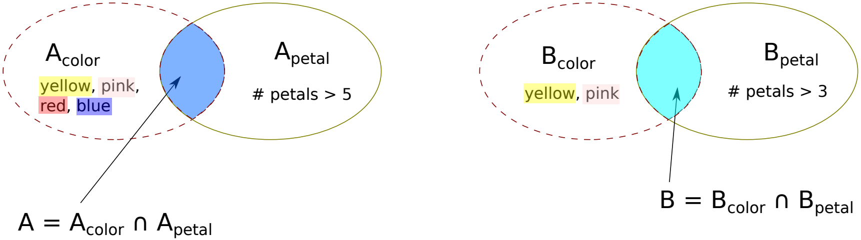
$\cdot r$ has the commutative and associative properties, i.e.,

$X \cdot_r Y = Y \cdot_r X$

$X \cdot_r Y \cdot_r Z = (X \cdot_r Y) \cdot_r Z = X \cdot_r (Y \cdot_r Z) \ where \ X, Y, Z \in \{0, 1, 2, 3, 4\}$

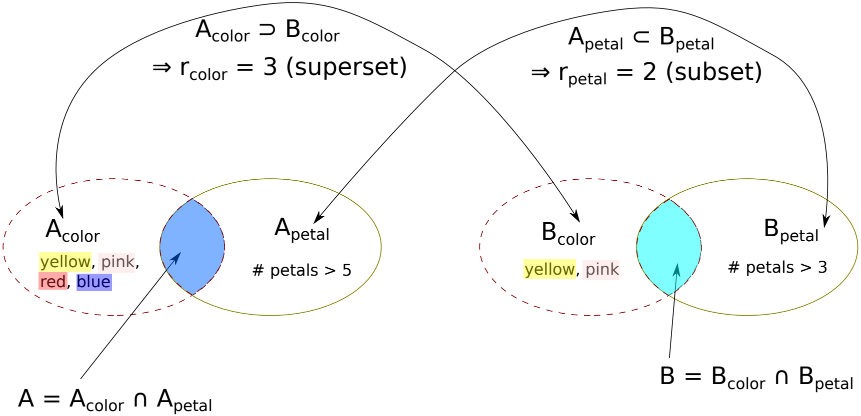
# Comparing Multi-Property Sets using $\cdot r$





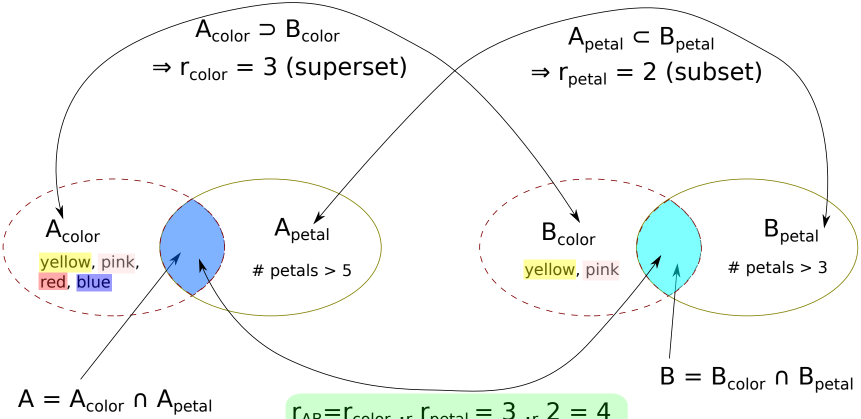
# Comparing Multi-Property Sets using $\cdot r$





# Comparing Multi-Property Sets using $\cdot_r$

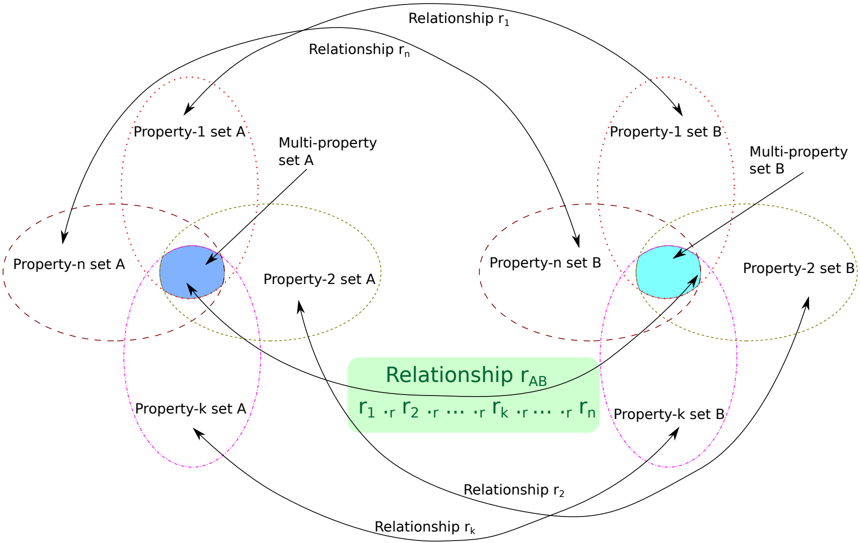





$r_{AB}=r_{color} \cdot_r r_{petal} = 3 \cdot_r 2 = 4$
$\Rightarrow$ A intersects B

# Comparing Multi-Property Sets using $\cdot_r$





# Normalizing SDN Rules by Matchmap, Actmap



Problem: diverse expressions of the match and action components of SDN rules complicate their automatic comparison based on multi-property set and $\cdot r$, e.g.,

rule 1's match: $\{$ *ip_src=192.168.1.1* , *tcp_dst=80* $\}$

rule 2's match: $\{$ *ip_dst=192.168.1.2* $\}$

# Normalizing SDN Rules by Matchmap, Actmap



Problem: diverse expressions of the match and action components of SDN rules complicate their automatic comparison based on multi-property set and $\cdot r$, e.g.,

rule 1's match: { *ip_src=192.168.1.1* , *tcp_dst=80* }

rule 2's match: { *ip_dst=192.168.1.2* }

Solution: normalizing the match and action components via a common template to obtain their uniform **matchmap** and **actmap**, e.g.,

| ip_src | ip_dst | tcp_dst |
|---|---|---|

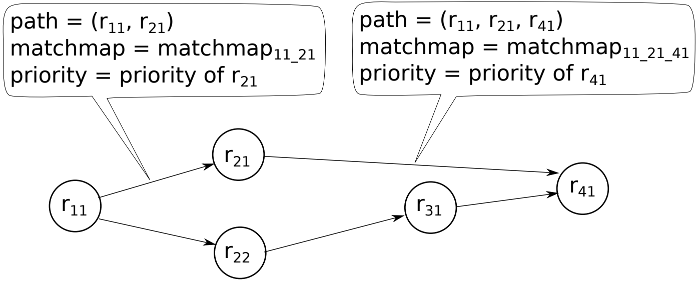
rule 1's **matchmap**: { *ip_src=192.168.1.1* , *ip_dst=any* , *tcp_dst=80* }

rule 2's **matchmap**: { *ip_src=any* , *ip_dst=192.168.1.2* , *tcp_dst=any* }

# Rule Graph

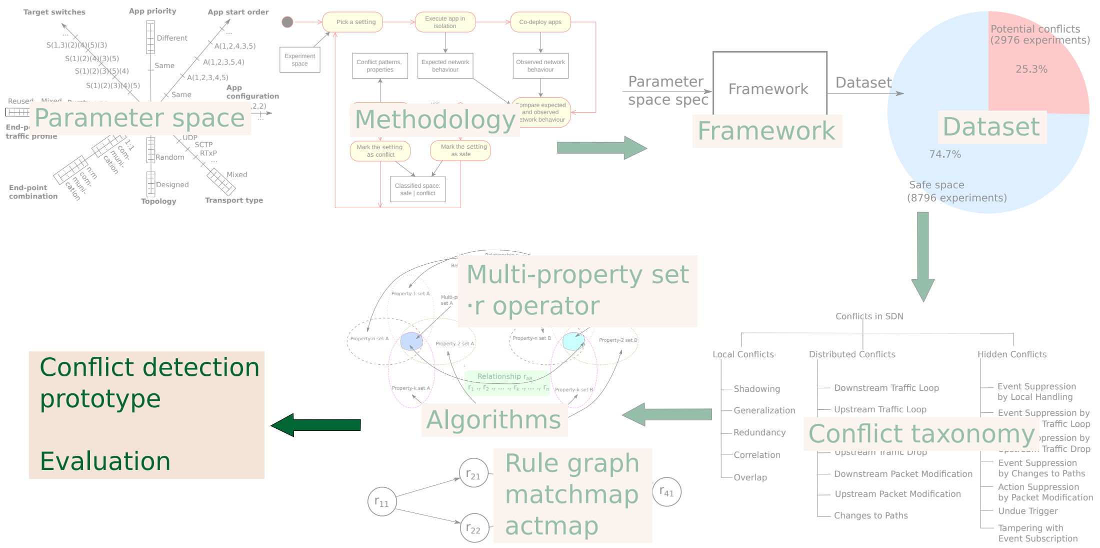


- A directed graph
- A vertex can represent a rule, an end-point, traffic drop or traffic loop



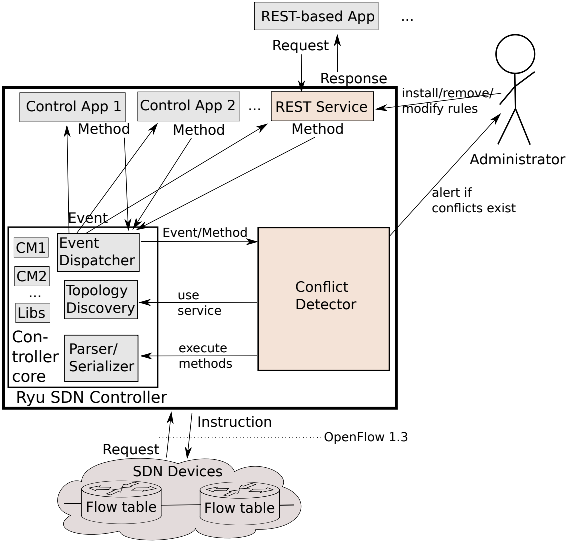

$r_{i\,j}$ : rule *i* in device *j*

# Proof-of-Concept



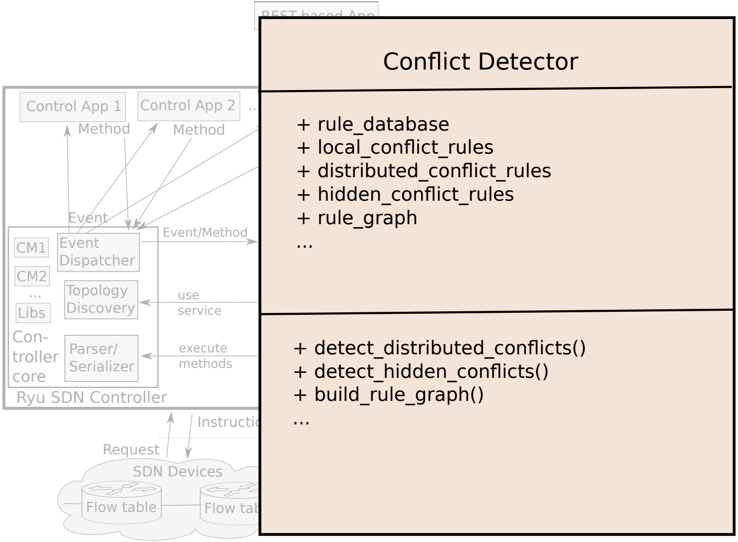
# Conflict Detection Prototype





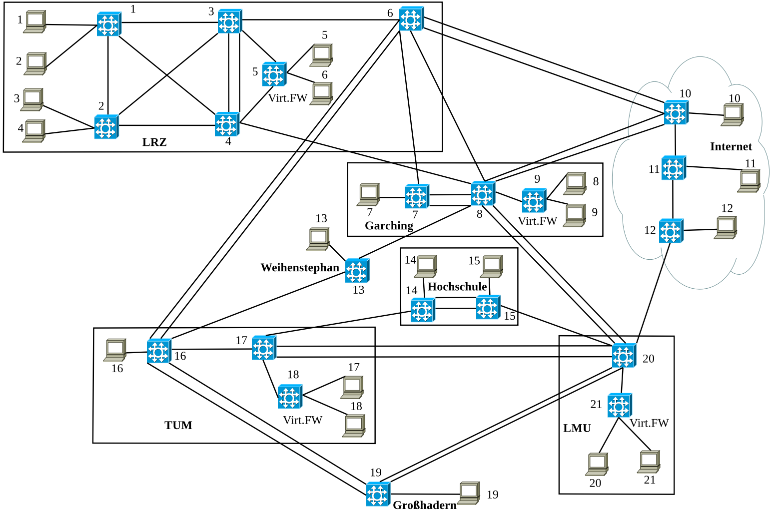
# Conflict Detection Prototype





# Network Topologies for Evaluation



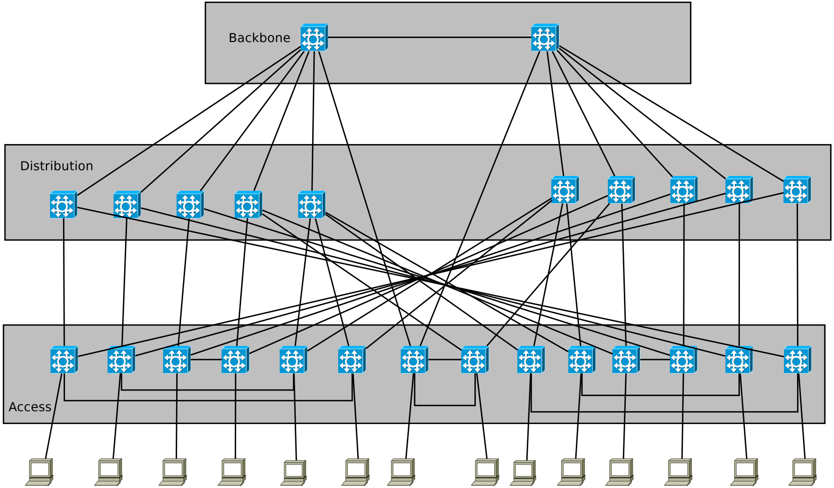



MWN topology:
(Münchner Wissenschaftsnetz)
21 switches,
21 end-points

https://www.lrz.de/services/netz/mwn-ueberblick/backbone.png

# Network Topologies for Evaluation






Stanford topology:
26 switches,
14 end-points

Kazemian, Peyman, George Varghese and Nick McKeown: *Header Space Analysis: Static Checking for Networks.* In NSDI, volume 12, pages 113–126, 2012

# Detected Results in Designed Cases

Rules are deployed with known conflicts
Conflicts detected by the prototype are then controlled manually

Results for both MWN and Stanford topologies:

| Test | Local conflicts | | | | | Traffic Loop | Traffic Drop | Hidden conflicts ESLH |
|---|---|---|---|---|---|---|---|---|
| | Shadowing | Generalization | Redundancy | Correlation | Overlap | | | |
| 1 | 1/1 | 1/1 | 1/1 | 1/1 | 1/1 | 1/1 | 1/1 | 1/1 |
| 2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 |
| 3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 |
| 4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 |
| 5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 |

detected by the prototype / designed

ESLH: Event Suppression by Local Handling

⇒ All conflicts are precisely identified

# Results in Randomly Checking Cases (MWN)



- The number of conflicts is unknown in advance
- Random conflict samples identified by the prototype are controlled manually

| Test | App Priority | # rules | Local conflicts | | | | | Traffic Loop | Traffic Drop | HC ESLH |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Sha | Gen | Red | Cor | Ove | | | |
| 1 | (2,2,2,2) | 790 | 0/0/0 | 0/0/0 | 0/0/0 | 27/10/10 | 0/0/0 | 0/0/0 | 0/0/0 | 60/10/10 |
| 2 | (2,2,3,4) | 803 | 0/0/0 | 0/0/0 | 0/0/0 | 26/10/10 | 0/0/0 | 0/0/0 | 0/0/0 | 60/10/10 |
| 3 | (3,2,2,3) | 816 | 0/0/0 | 0/0/0 | 0/0/0 | 27/10/10 | 0/0/0 | 0/0/0 | 0/0/0 | 60/10/10 |
| 4 | (3,5,2,4) | 789 | 0/0/0 | 0/0/0 | 0/0/0 | 25/10/10 | 0/0/0 | 0/0/0 | 0/0/0 | 59/10/10 |
| 5 | (5,4,3,2) | 791 | 0/0/0 | 0/0/0 | 0/0/0 | 24/10/10 | 0/0/0 | 0/0/0 | 0/0/0 | 60/10/10 |

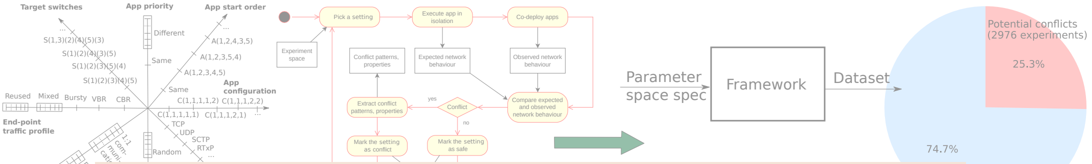
Sha: Shadowing
Gen: Generalization
Red: Redundancy
Cor: Correlation, Ove: Overlap

detected by the prototype/
randomly selected/
confirmed via manual control

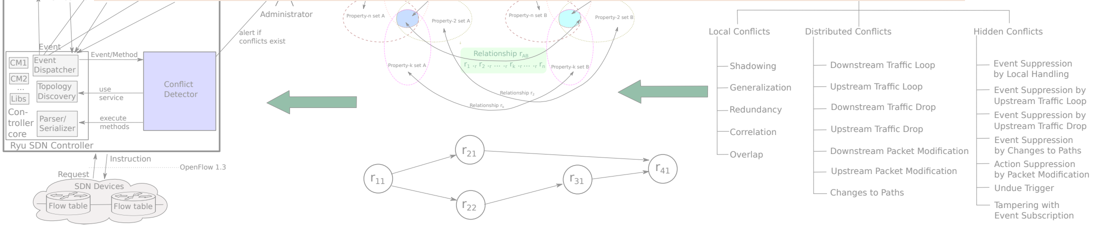
HC ESLH:
Hidden Conflict
Event Suppression
by Local Handling

⇒ All randomly checking conflicts are correct

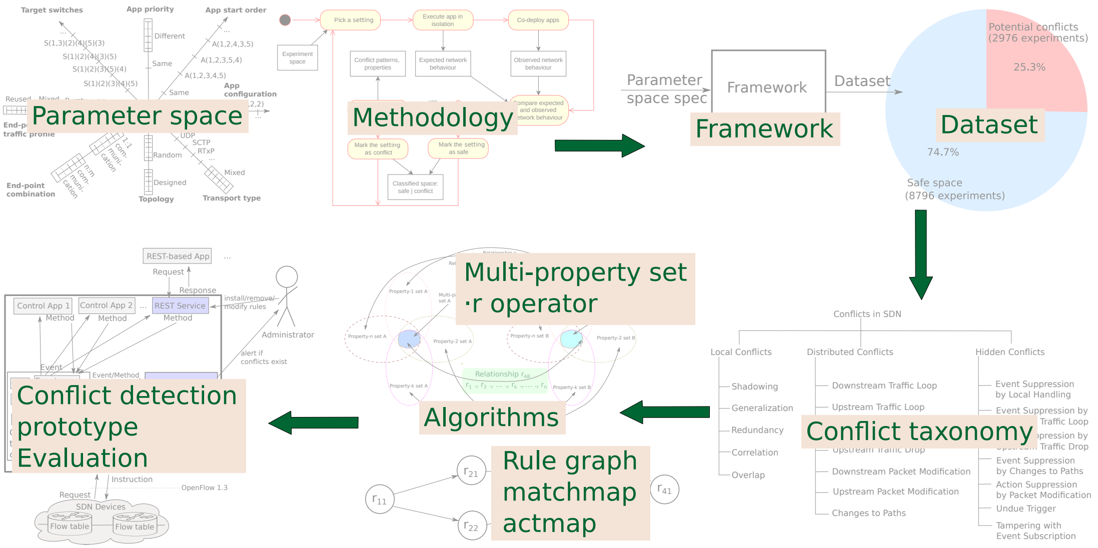
# Conclusions




RQ1: A suitable method for researching conflicts in SDN
RQ2: Conflict classification
RQ3: Conflict detection



# Conclusions



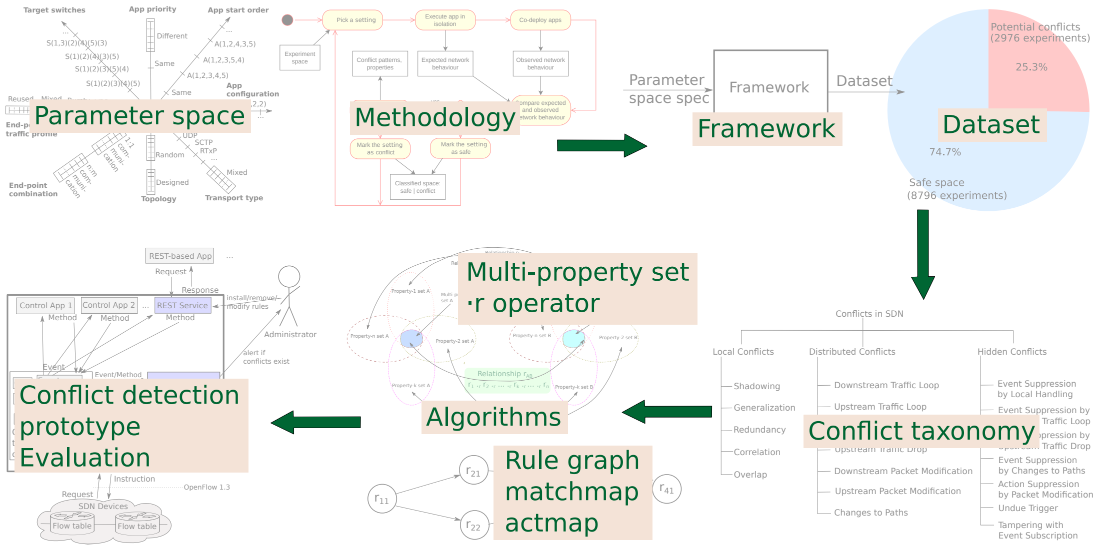
# Prospects



- SDN technologies as a new dimension: OpenFlow, POF, P4 and P4Runtime
- Topology changes
- Matching policies: *first match*, *best match*, *deny take precedence*, *most/least specific take precedence*
- Real-time conflict detection
- Conflict resolution
- Conflict avoidance

# Conclusions

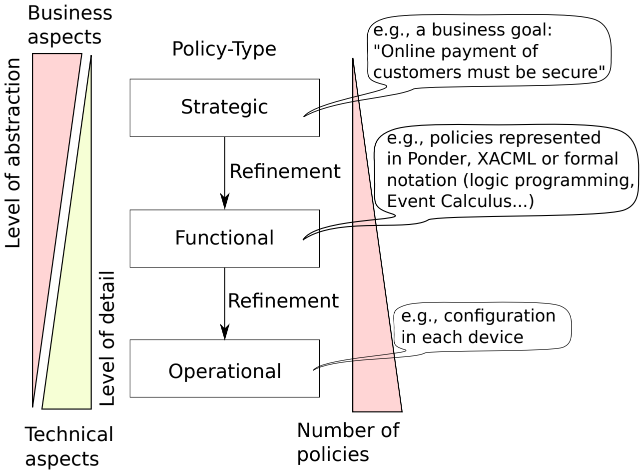




# Backup

# Policy Hierarchy

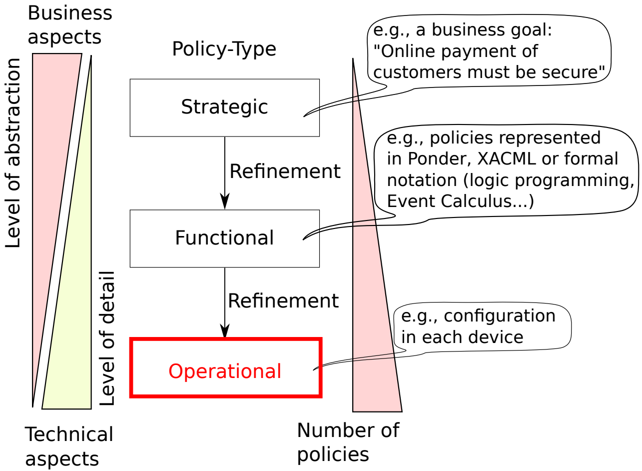





Danciu, Vitalian: *Application of policy-based techniques to process-oriented IT service management.* PhD thesis, LMU, 2007

# Policy Hierarchy

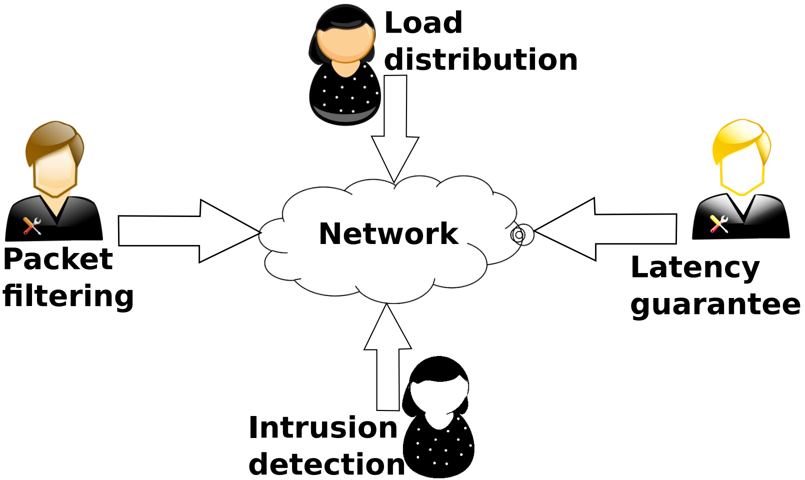





Danciu, Vitalian: *Application of policy-based techniques to process-oriented IT service management.* PhD thesis, LMU, 2007

# Conflict in Network Management





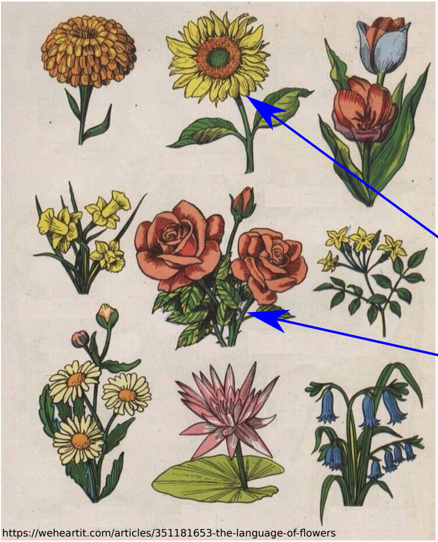
# Goal - Analogy

https://weheartit.com/articles/351181653-the-language-of-flowers

Q: How to identify these flowers?

# Goal - Analogy






Q: How to identify these flowers?

A: Based on their patterns, properties

disk-shaped, yellow, sun-facing flowers

sharp prickle stems, showy red fragrant flowers

# Goal - Analogy



https://weheartit.com/articles/351181653-the-language-of-flowers

Q: How to identify these flowers?

How many kinds? (classification)

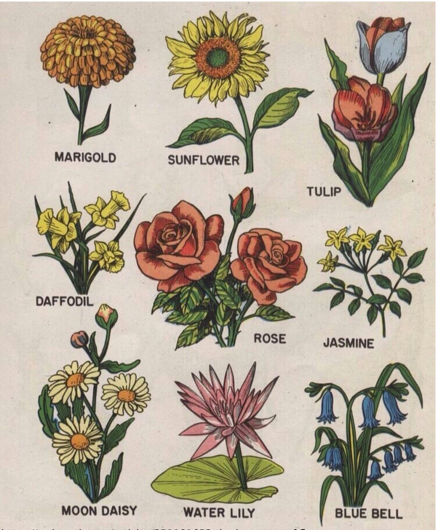
Where can they be found? (localization)

Which research methods are appropriate?

...

# Goal - Analogy






https://weheartit.com/articles/351181653-the-language-of-flowers

https://weheartit.com/entry/336964443

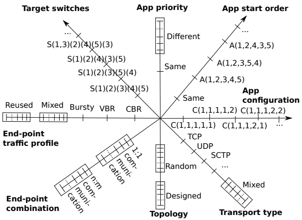

https://www.amazon.in/Parag-Fragrances-Rose-Perfume-Women/dp/B07LFXM28L

# Space for Experiments

- $t$ different topologies
- $x$ transport protocols and their combinations
- $a$ applications, each has maximal $c$ configurations
- $s$ switches in the topology
- $p$ traffic profiles
- $e$ end-points get involved in the test

Consequently, there are

- $t$ points on the *Topology* axis,
- $x$ points on the *Transport type* axis,
- $A = \sum_{i=2}^{a} c^i$ points on the *App. configuration* axis,
- $O = \sum_{j=2}^{a} \binom{a}{j} \times j!$ points on the *App. start order* axis,
- $P = \sum_{k=2}^{a} (k^k - k + 1)$ values on the *App. priority* axis,
- $S = \sum_{l=2}^{a} (2^s - 1)^l$ points on the *Target switches* axis,
- $p$ points on the *End-point traffic profile* axis,
- $C = \sum_{m=2}^{e} \binom{e}{m} \cdot m!$ points on the *End-point combination* axis.




Example:

$t = 5$ topos, $s = 10$ switches each,
$a = 5$ apps, $c = 2$ configs each,
$x = 2$ transport types (TCP/UDP),
$p = 5$ traffic profiles,
$e = 5$ end-points involved in each test

$\Rightarrow 10^{27}$ points (or experimental settings)

$\Rightarrow$ more than $3 \cdot 10^{16}$ years if each experiment takes 1ms!

$$\Rightarrow \Omega = t \cdot x \cdot A \cdot O \cdot P \cdot S \cdot p \cdot C = t \cdot x \cdot \sum_{i=2}^{a} c^i \cdot \sum_{j=2}^{a} \binom{a}{j} \cdot j! \cdot \sum_{k=2}^{a} (k^k - k + 1) \cdot \sum_{l=2}^{a} (2^s - 1)^l \cdot p \cdot \sum_{m=2}^{e} \binom{e}{m} \cdot m!$$

# Control applications for Experiments

| Apps | Active | Passive | Controller built-in | Restful | Target traffic |
|---|---|---|---|---|---|
| ARP cache | ✓ | ✓ | ✓ | | ARP |
| Routing | | ✓ | ✓ | | ARP, ICMP, TCP, UDP |
| EpLB | | ✓ | ✓ | | TCP, UDP |
| PLB | ✓ | | ✓ | | TCP, UDP |
| PPLB4S | | ✓ | ✓ | | TCP, UDP |
| PPLB4D | | ✓ | ✓ | | TCP, UDP |
| Firewall | ✓ | | ✓ | ✓ | TCP, UDP |
| TE | ✓ | | ✓ | ✓ | TCP, UDP |

EpLB: End-point Load Balancer, PLB: Active Path Load Balancer, PPLB4S: Source-based Passive Path Load Balancer, PPLB4D: Destination-based Passive Path Load Balancer, TE: Traffic Engineering

# Compact Subspaces for Experiments

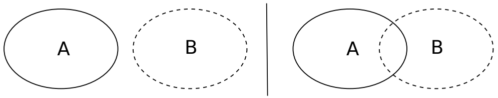


- Exploiting applications' characteristics, e.g., deploying FW at the network boundary
- Pragmatically favouring points where conflicts are possible
- non-redundant points
- valid points

# Compact Subspaces for Experiments



- Exploiting applications' characteristics, e.g., deploying FW at the network boundary
- Pragmatically favouring points where conflicts are possible
- non-redundant points
- valid points

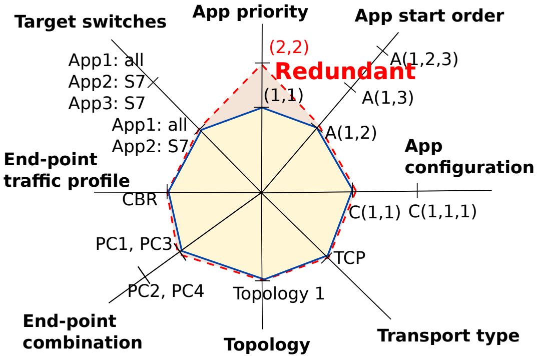



**More favourable**

# Compact Subspaces for Experiments



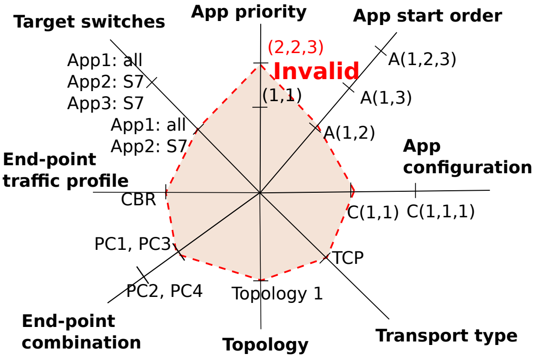
- Exploiting applications' characteristics, e.g., deploying FW at the network boundary
- Pragmatically favouring points where conflicts are possible
- non-redundant points
- valid points



# Compact Subspaces for Experiments



- Exploiting applications' characteristics, e.g., deploying FW at the network boundary
- Pragmatically favouring points where conflicts are possible
- non-redundant points
- valid points

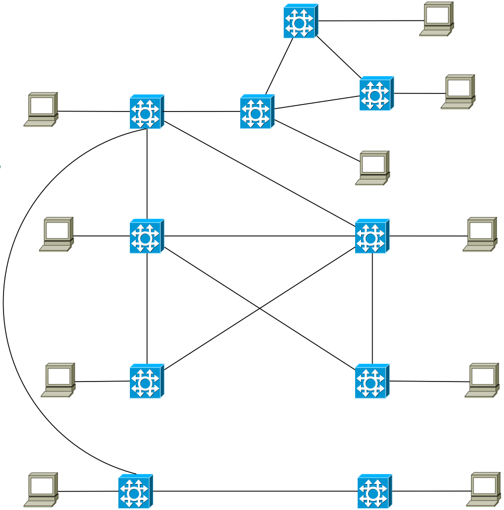


# Explored Subspaces

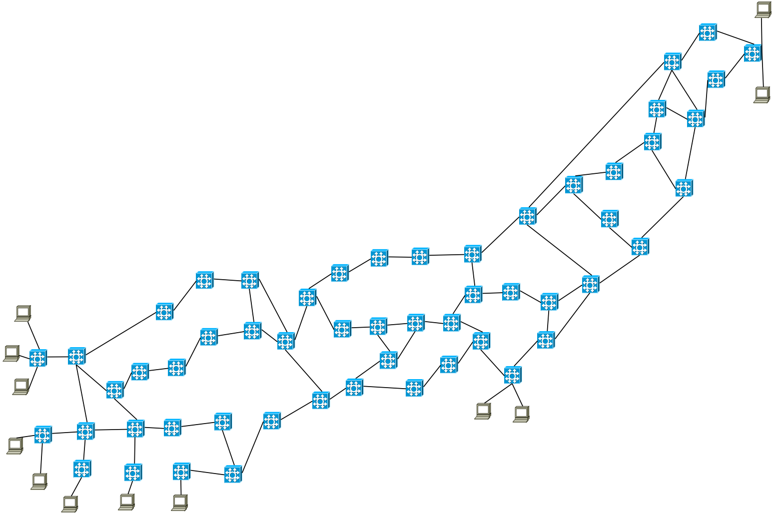
| Category | Value | Note |
|---|---|---|
| # Topologies | 12 | 6 designed topologies, 6 random topologies, # end-points ranges from 4 to 21, # switches from 3 to 55 |
| # Applications | 14 | containing fundamental functions (e.g., topology discovery, ARP cache, NDP cache) and applications involved directly in conflict study, e.g., End-point Load Balancer, Path Load Balancer, Fire-wall, Path Enforcer. . . |
| App. configuration | $1 \rightarrow 5$ | each app. has at least 1 configuration, at most 5 |
| App. start order | same and different | at least two apps. are co-deployed in an experiment, at most 5 |
| App. priority | same and different | the co-deployment of 2 apps. yields 3 combinations of priority, there are 541 combinations for 5 apps |
| Target switches | $1 \rightarrow$ all | each app. can have one target switch or more, or even deploy its rules on all switches, e.g., the Shortest Path First app |
| Ep. Traffic Profile | CBR and VBR | *netcat* and *iperf* programs are used to generate TCP/UDP traffic |
| EP. Combination | unicast, multicast | multicast traffic is generated for the MEADcast app. in IPv6, all other apps. are active on IPv4 unicast traffic |
| Transport type | TCP, UDP | - |
| # Experiments | 11,772 | 8796 experiments expose no conflict, 2976 experiments show po-tential conflicts (these experiments are conducted automatically, the manual experiments are not counted) |

# Network topologies

A designed network topology with 10 switches and 10 hosts



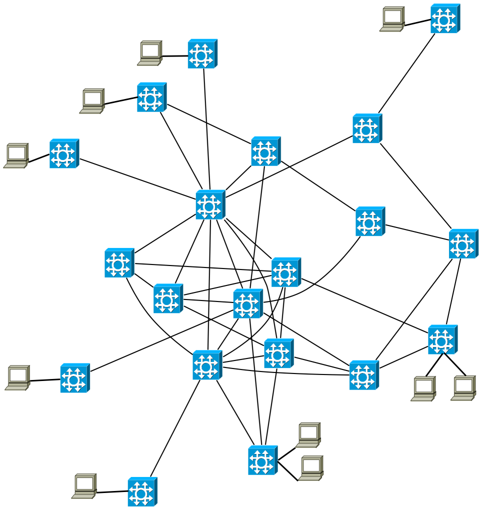
A designed topology simulating the core backbone of the Nippon Telegraph and Telephone (NTT) network in Japan (55 switches, 12 hosts)



# Network topologies

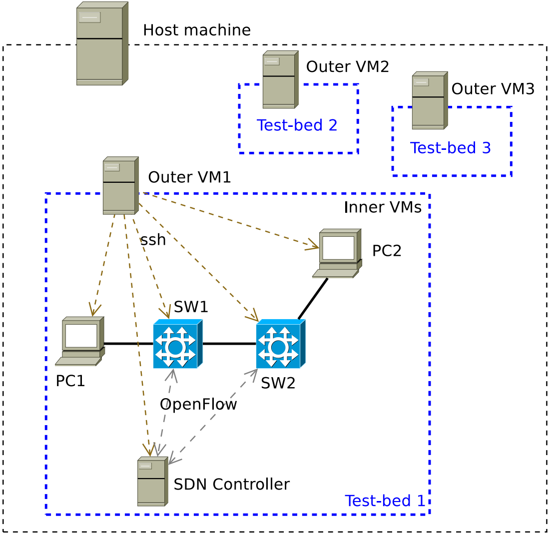


A random topology with
20 switches and 10 hosts

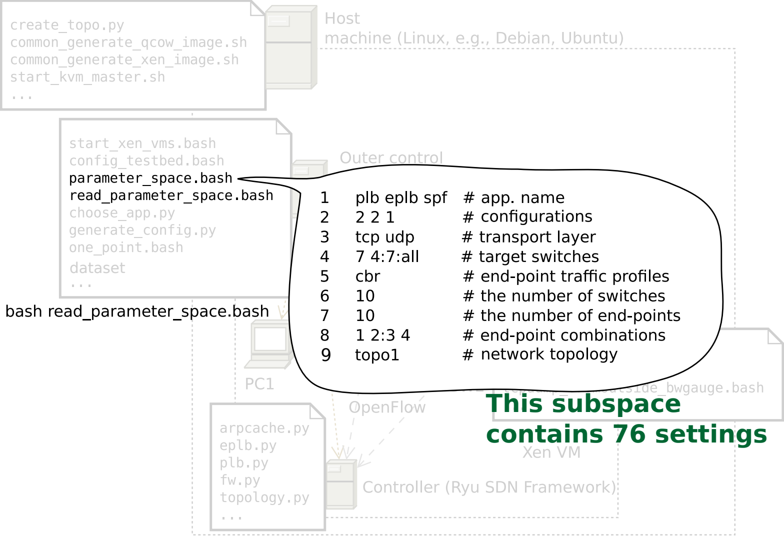


# Framework for automating experiments (short)





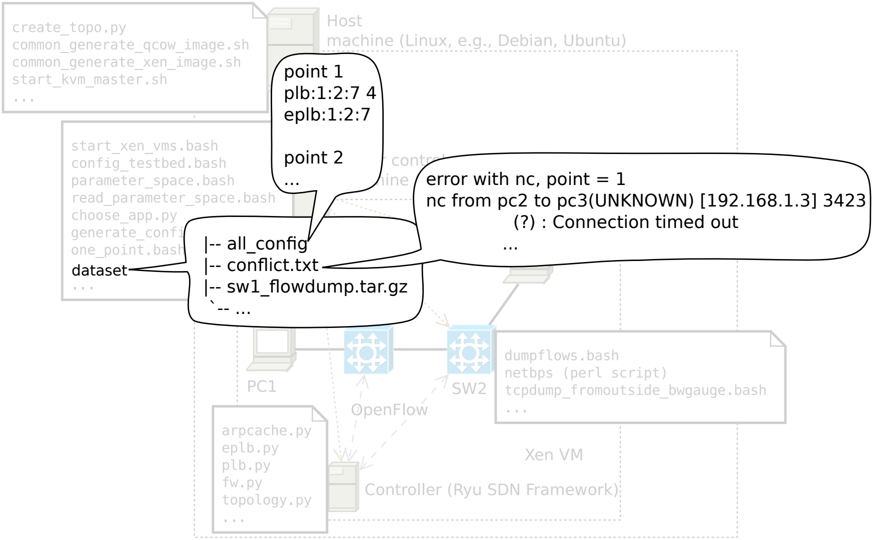
# Framework for automating experiments (short)




**This subspace contains 76 settings**

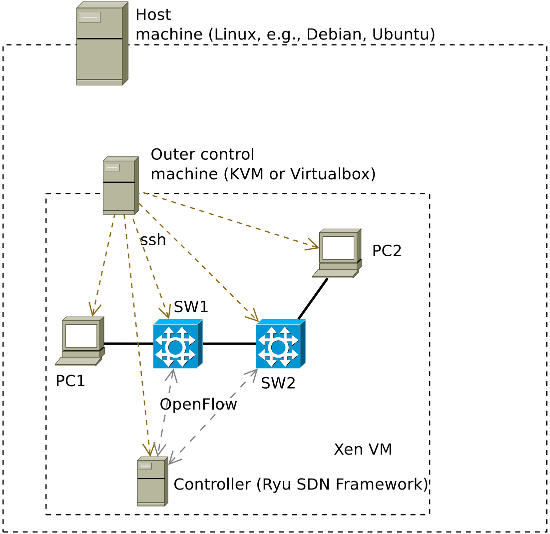
# Framework for automating experiments (short)





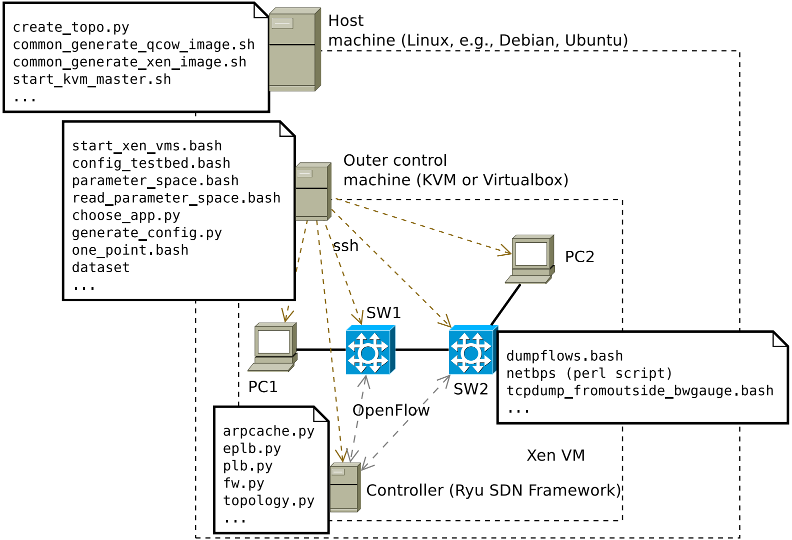
# Framework for automating experiments (long)





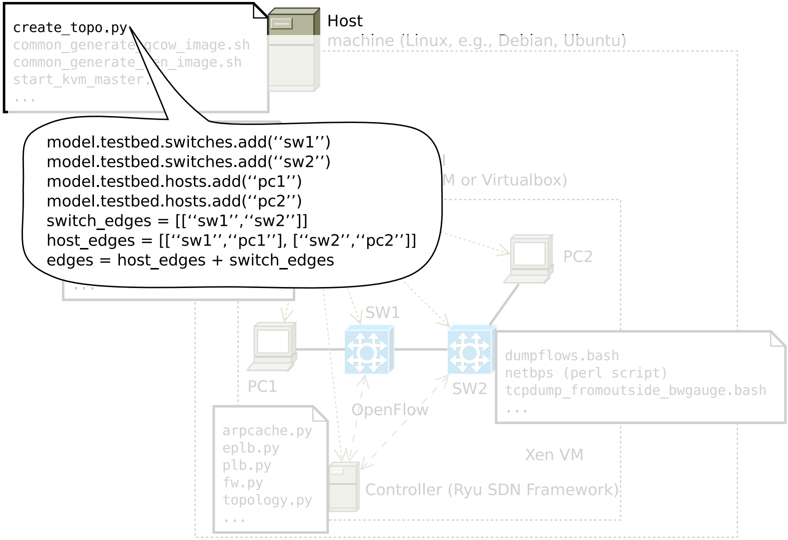
# Framework for automating experiments (long)





# Framework for automating experiments (long)





# Framework for automating experiments (long)

create_topo.py
common_generate_qcow_image.sh
common_generate_xen_image.sh
start_kvm_master.sh
...

Host
machine (Linux, e.g., Debian, Ubuntu)

start_xen_vms.bash
config_testbed.bash
parameter_space.bash
read_parameter_space.bash
choose_app.py
generate_config.py
one_point.bash
dataset
...

Outer control

bash read_parameter_space.bash

| | | |
|---|---|---|
| 1 | plb eplb spf | # app. name, e.g., Path Load Balancer |
| 2 | 2 2 1 | # the number of configurations of each app. in line 1 |
| 3 | tcp udp | # transport layer |
| 4 | 7 4:7:all | # target switches |
| 5 | cbr | # end-point traffic profiles |
| 6 | 10 | # the number of switches in the network topology |
| 7 | 10 | # the number of end-points in the network topology |
| 8 | 1 2:3 4 | # end-point combinations, sources : targets |
| 9 | topo1 | # name of the tested network topology |

PC1

OpenFlow

...

**This subspace contains 76 settings**

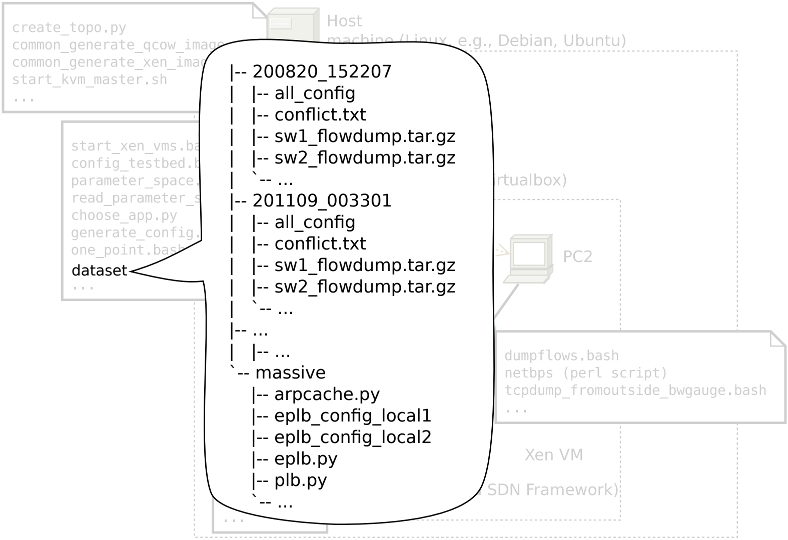
arpcache.py
eplb.py
plb.py
fw.py
topology.py
...

Xen VM

Controller (Ryu SDN Framework)

# Framework for automating experiments (long)





# Framework for automating experiments (long)



dataset

```
|-- 200820_152207
|   |-- all_config
|   |-- conflict.txt
|   |-- sw1_flowdump.tar.gz
|   |-- sw2_flowdump.tar.gz
|   `-- ...
|-- 201109_003301
|   |-- all_config
|   |-- conflict.txt
|   |-- sw1_flowdump.tar.gz
|   |-- sw2_flowdump.tar.gz
|   `-- ...
|-- ...
|   |-- ...
`-- massive
    |-- arpcache.py
    |-- eplb_config_local1
    |-- eplb_config_local2
    |-- eplb.py
    |-- plb.py
    `-- ...
```

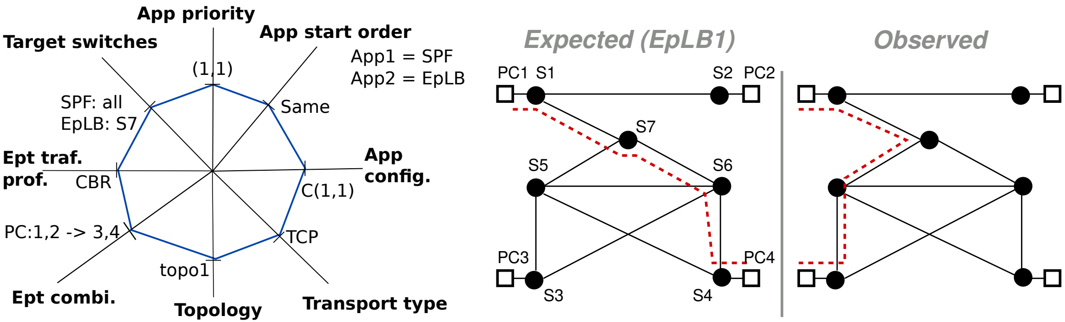
all_config:

```
point 1
plb:1:2:7 4
eplb:1:2:7
point 2
...
point 76
plb:2:4:7 4
eplb:2:3:7
spf:1:2:all
```

conflict.txt:

```
error with nc, point = 1
nc from pc2 to pc3(UNKNOWN) [192.168.1.3] 3423
        (?) : Connection timed out
error with iperf, point = 1, iperf from pc2 to pc3
[ 3] WARNING: did not receive ack of last datagram
        after 10 tries.
...
```

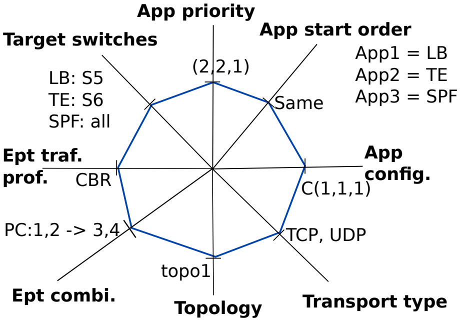
# Example 1: Analysis of Experiments

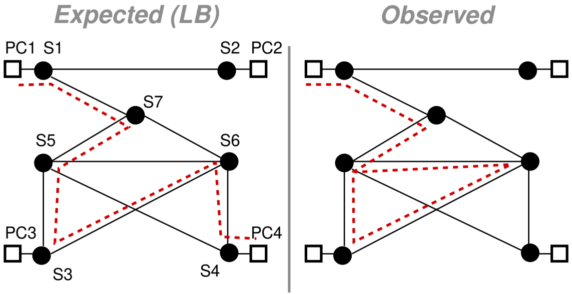



## Conflict pattern:

$$Correlation : priority_i = priority_j, \ match_i \subseteq match_j \ \vee \ match_i \supseteq match_j, action_i \neq action_j$$

# Example 2: Analysis of Experiments




Conflict property:

Traffic loop: target traffic of a control application is caught in a loop

# Conflict Detection



Conflicts in SDN

- Local Conflicts
  - Shadowing
  - Generalization
  - Redundancy
  - Correlation
  - Overlap
- Distributed Conflicts
  - Downstream Traffic Loop
  - Upstream Traffic Loop
  - Downstream Traffic Drop
  - Upstream Traffic Drop
  - Downstream Packet Modification
  - Upstream Packet Modification
  - Changes to Paths
- Hidden Conflicts
  - Event Suppression by Local Handling
  - Event Suppression by Upstream Traffic Loop
  - Event Suppression by Upstream Traffic Drop
  - Event Suppression by Changes to Paths
  - Action Suppression by Packet Modification
  - Undue Trigger
  - Tampering with Event Subscription

detected by employing the multi-property set concepts, relationship combination operator .r, matchmap, actmap

# Conflict Detection



Conflicts in SDN

- Local Conflicts
  - Shadowing
  - Redundancy
  - Correlation
- Distributed Conflicts
  - Downstream Traffic Loop
  - Upstream Traffic Loop
  - Downstream Traffic Drop
  - Upstream Traffic Drop
  - Downstream Packet Modification
  - Upstream Packet Modification
  - Changes to Paths
- Hidden Conflicts
  - Event Suppression by Local Handling
  - Event Suppression by Upstream Traffic Loop
  - Event Suppression by Upstream Traffic Drop
  - Event Suppression by Changes to Paths
  - Action Suppression by Packet Modification
  - Undue Trigger
  - Tampering with Event Subscription

handled by employing the rule graph to detect general traffic loop, traffic drop and packet modification

# Conflict Detection

## Conflicts in SDN

- Local Conflicts
  - Shadowing
  - Generalization
  - Redundancy
  - Correlation
  - Overlap
- Distributed Conflicts
  - Downstream Traffic Loop
  - Upstream Traffic Loop
  - Downstream Traffic Drop
  - Upstream Traffic Drop
  - Downstream Packet Modification
  - Upstream Packet Modification
  - Changes to Paths
- Hidden Conflicts
  - Event Suppression by Local Handling
  - Event Suppression by Upstream Traffic Loop
  - Event Suppression by Upstream Traffic Drop
  - Event Suppression by Changes to Paths
  - Action Suppression by Packet Modification
  - Undue Trigger
  - Tampering with Event Subscription

avoided by choosing the target switches for the control applications such that all possible traffic paths between sources and destinations always contain these switches

# Conflict Detection

Conflicts in SDN

- Local Conflicts
  - Shadowing
  - Generalization
  - Redundancy
  - Correlation
  - Overlap
- Distributed Conflicts
  - Downstream Traffic Loop
  - Upstream Traffic Loop
  - Downstream Traffic Drop
  - Upstream Traffic Drop
  - Downstream Packet Modification
  - Upstream Packet Modification
  - Changes to Paths
- Hidden Conflicts
  - Event Suppression by Local Handling
  - Event Suppression by Upstream Traffic Loop
  - Event Suppression by Upstream Traffic Drop
  - Event Suppression by Changes to Paths
  - Action Suppression by Packet Modification
  - Undue Trigger
  - Tampering with Event Subscription

detected based on
input from control applications,
and by employing the
multi-property set concepts,
relationship combination operator .r,
matchmap, actmap

# Conflict Detection

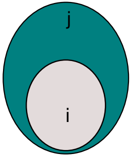


Conflicts in SDN

- Local Conflicts
  - Shadowing
  - Generalization
  - Redundancy
  - Correlation
  - Overlap
- Distributed Conflicts
  - Downstream Traffic Loop
  - Upstream Traffic Loop
  - Downstream Traffic Drop
  - Upstream Traffic Drop
  - Downstream Packet Modification
  - Upstream Packet Modification
  - Changes to Paths
- Hidden Conflicts
  - Event Suppression by Local Handling
  - Event Suppression by Upstream Traffic Loop
  - Event Suppression by Upstream Traffic Drop
  - Event Suppression by Changes to Paths
  - Action Suppression by Packet Modification
  - Undue Trigger
  - Tampering with Event Subscription

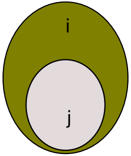
Coped with by securing the control plane using existing solutions from related work (out of cope)

# Local Conflicts




Action i ≠ Action j ⇒ Shadowing

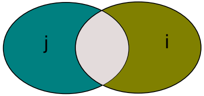
Action i = Action j ⇒ Redundancy




Action i ≠ Action j ⇒ Generalization

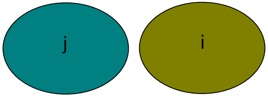
Action i = Action j ⇒ Overlap1

**priority i < priority j**




Action i ≠ Action j ⇒ Correlation2

Action i = Action j ⇒ Overlap2

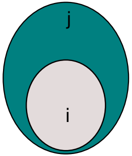



Any action ⇒ No conflict

Legend:

- Match space of rule i
- Match space of rule j
- Intersection of the match spaces of rules i and j

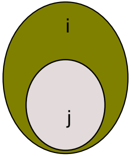
# Local Conflicts




Action i ≠ Action j ⇒ Shadowing

Action i = Action j ⇒ Redundancy

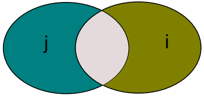
Rule i: pri = 2, src=192.168.1.**1**, dst=192.168.1.**2**, action=outport:1
Rule j: pri = 3, src=192.168.1.**1**, dst= **any**, action=outport:2




Action i ≠ Action j ⇒ Generalization

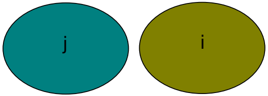
Action i = Action j ⇒ Overlap1

**priority i < priority j**




Action i ≠ Action j ⇒ Correlation2

Action i = Action j ⇒ Overlap2



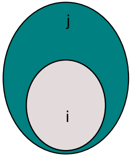

Any action ⇒ No conflict

Legend:

- Match space of rule i
- Match space of rule j
- Intersection of the match spaces of rules i and j

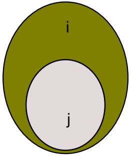
# Local Conflicts





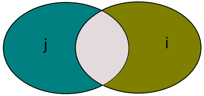

Action i ≠ Action j ⇒ Correlation1

Action i = Action j ⇒ Redundancy



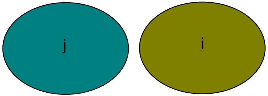

Action i ≠ Action j ⇒ Correlation1

Action i = Action j ⇒ Redundancy




Action i ≠ Action j ⇒ Correlation2

Action i = Action j ⇒ Overlap2




Any action ⇒ No conflict

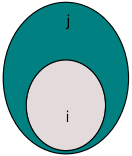
**priority i = priority j**

Legend:
- Match space of rule i
- Match space of rule j
- Intersection of the match spaces of rules i and j

# Local Conflicts

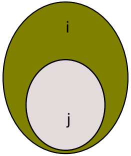


Rule i: pri = 2, src=192.168.1.**1**, dst=192.168.1.**2**, action=outport:1
Rule j: pri = 2, src=192.168.1.**1**, dst= **any**, action=outport:2



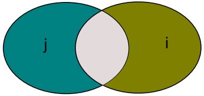

Action i ≠ Action j ⇒ Correlation1

Action i = Action j ⇒ Redundancy



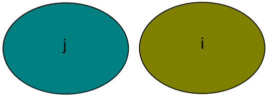

Action i ≠ Action j ⇒ Correlation1

Action i = Action j ⇒ Redundancy




Action i ≠ Action j ⇒ Correlation2

Action i = Action j ⇒ Overlap2




Any action ⇒ No conflict

**priority i = priority j**

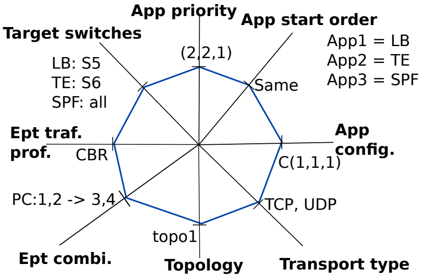
Legend:
- Match space of rule i
- Match space of rule j
- Intersection of the match spaces of rules i and j

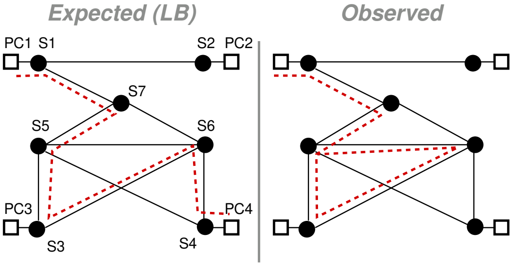
# Distributed Conflicts

| Cause \ Direction | downstream | upstream |
|---|---|---|
| Traffic loop | ✓ | ✓ |
| Traffic drop | ✓ | ✓ |
| Packet modification | ✓ | ✓ |
| Changes to paths | ✗ | ✓ |

# Example: Distributed Conflicts

1. Downstream traffic loop
2. Upstream traffic loop
3. Downstream traffic drop
4. Upstream traffic drop
5. Downstream packet modification
6. Upstream packet modification
7. Changes to paths

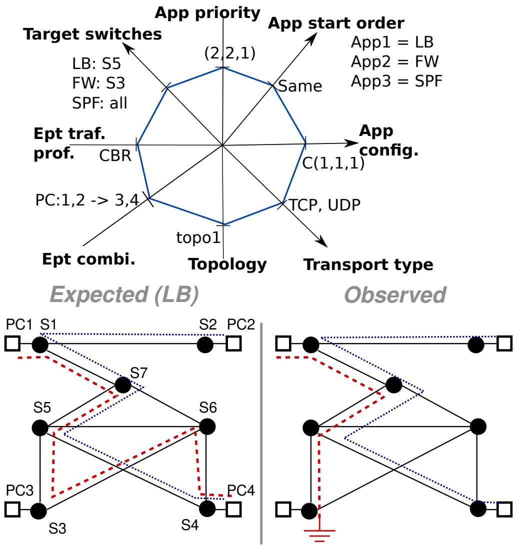


# Example: Distributed Conflicts

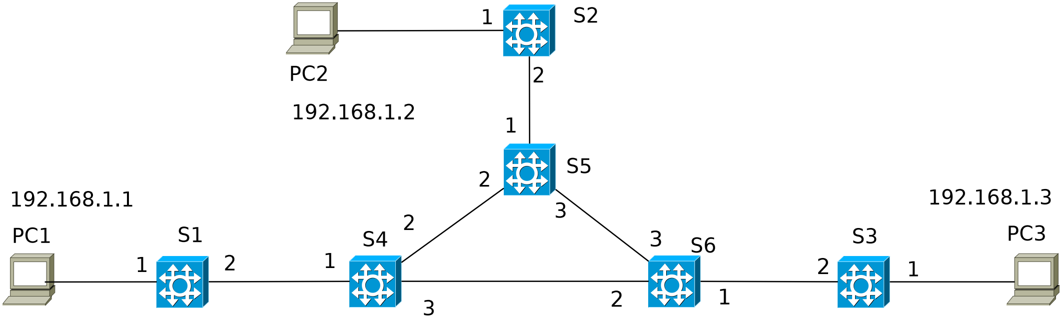


1. Downstream traffic loop
2. Upstream traffic loop
3. Downstream traffic drop
4. Upstream traffic drop
5. Downstream packet modification
6. Upstream packet modification
7. Changes to paths



# Distributed Conflicts



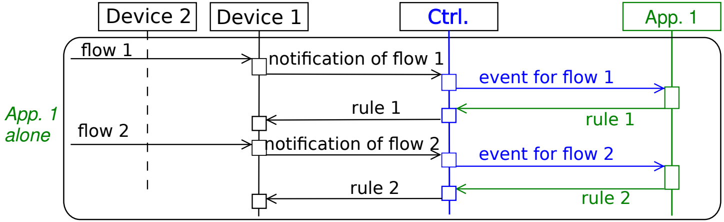



- Traffic black hole
- Traffic loop

E.g., S4: src=pc1, out:2 S5: src=pc1, out:3 S6: src=pc1,out:2

# Example: Hidden Conflicts






rule : <match, action>

Example:

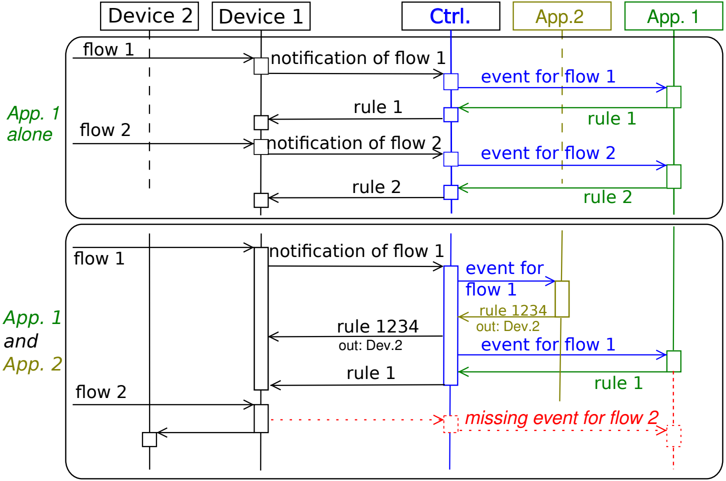
rule 1: <src = 192.168.1.**1**, dst = 192.168.1.**2**, action=port **1**>

rule 2: <src = 192.168.1.**1**, dst = 192.168.1.**3**, action=port **1**>

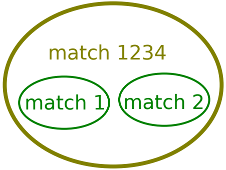
rule 1234: <src = 192.168.1.**1**, dst = **any**, action=port **2**>

# Example: Hidden Conflicts



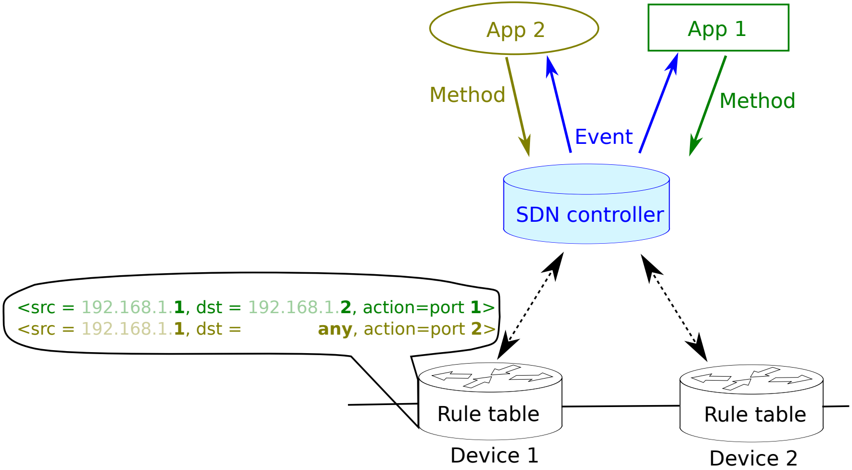



rule : <match, action>



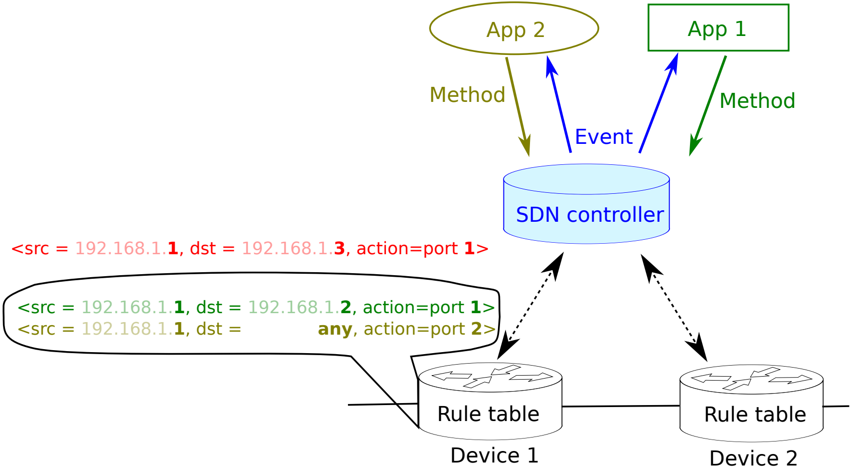
# Example: Hidden Conflicts





# Example: Hidden Conflicts





# Hidden Conflicts



**Properties:**

- Causes are hidden, rule tables alone reveal no (or some different) problem
- Insight into the mechanics of the control plane is necessary to identify the causes

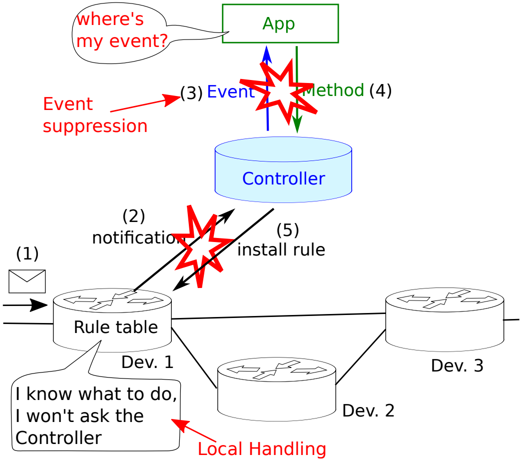
**Consequences:**

- Suppression of events
- Application failure

# Hidden Conflicts' Causes

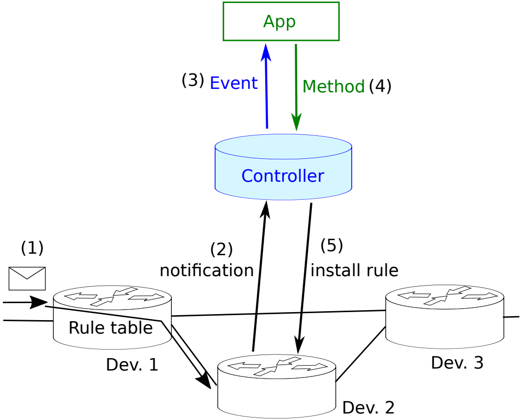


1. Event suppression by local handling
2. Event suppression by upstream traffic loop
3. Event suppression by upstream traffic drop
4. Event suppression by changes to paths
5. Action suppression by modification of packets
6. Undue trigger
7. Tampering with event subscription



# Hidden Conflicts' Causes

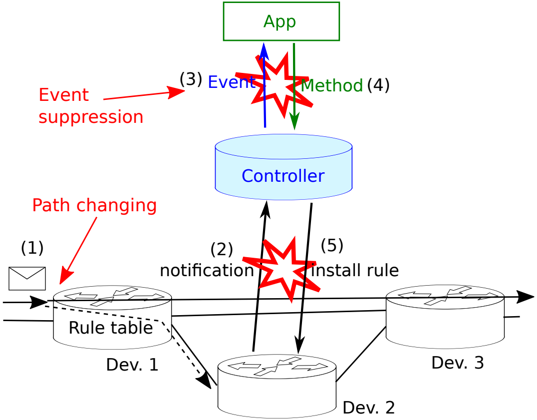
1. Event suppression by local handling
2. Event suppression by upstream traffic loop
3. Event suppression by upstream traffic drop
4. Event suppression by changes to paths
5. Action suppression by modification of packets
6. Undue trigger
7. Tampering with event subscription



# Hidden Conflicts' Causes

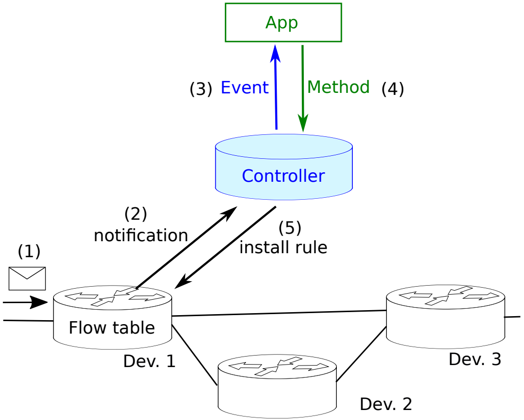


1. Event suppression by local handling
2. Event suppression by upstream traffic loop
3. Event suppression by upstream traffic drop
4. **Event suppression by changes to paths**
5. Action suppression by modification of packets
6. Undue trigger
7. Tampering with event subscription



# Hidden Conflicts' Causes

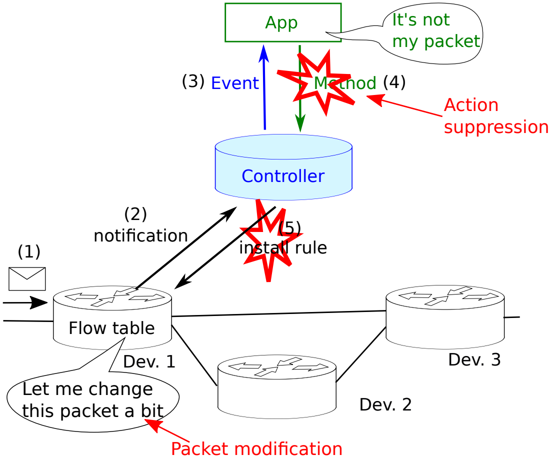
1. Event suppression by local handling
2. Event suppression by upstream traffic loop
3. Event suppression by upstream traffic drop
4. Event suppression by changes to paths
5. **Action suppression by modification of packets**
6. Undue trigger
7. Tampering with event subscription



# Hidden Conflicts' Causes



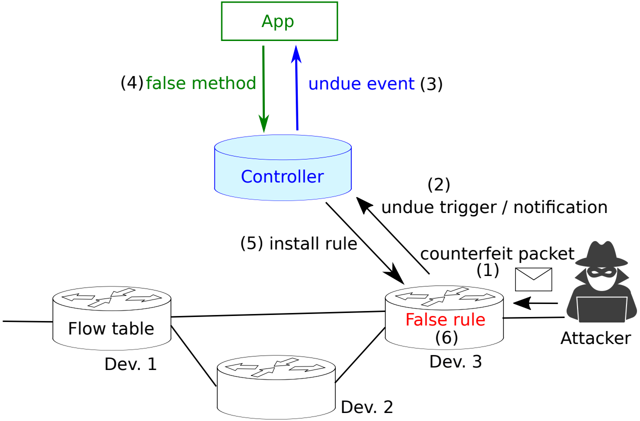
1. Event suppression by local handling
2. Event suppression by upstream traffic loop
3. Event suppression by upstream traffic drop
4. Event suppression by changes to paths
5. Action suppression by modification of packets
6. Undue trigger
7. Tampering with event subscription



# Hidden Conflicts' Causes

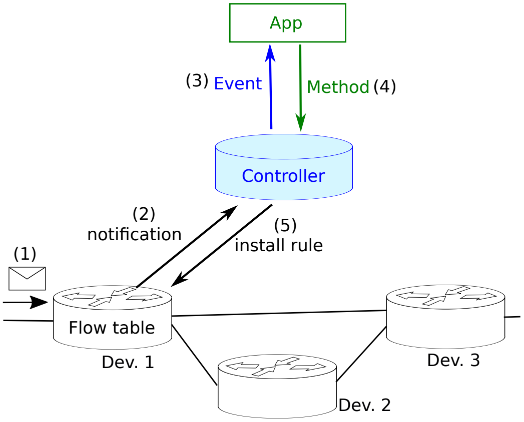


1. Event suppression by local handling
2. Event suppression by upstream traffic loop
3. Event suppression by upstream traffic drop
4. Event suppression by changes to paths
5. Action suppression by modification of packets
6. Undue trigger
7. Tampering with event subscription



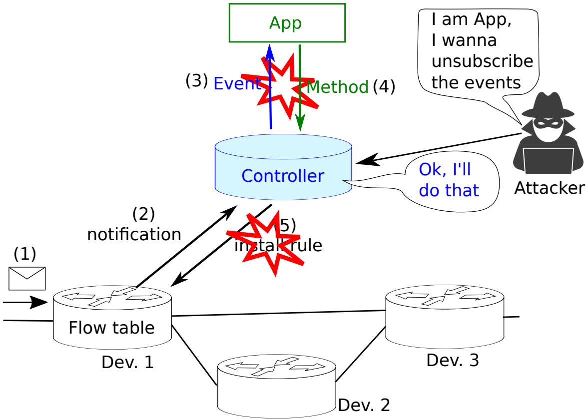
# Hidden Conflicts' Causes

1. Event suppression by local handling
2. Event suppression by upstream traffic loop
3. Event suppression by upstream traffic drop
4. Event suppression by changes to paths
5. Action suppression by modification of packets
6. Undue trigger
7. Tampering with event subscription



# Hidden Conflicts' Causes

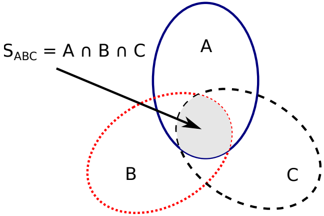
1. Event suppression by local handling
2. Event suppression by upstream traffic loop
3. Event suppression by upstream traffic drop
4. Event suppression by changes to paths
5. Action suppression by modification of packets
6. Undue trigger
7. Tampering with event subscription



# Overview of the Measures (1)



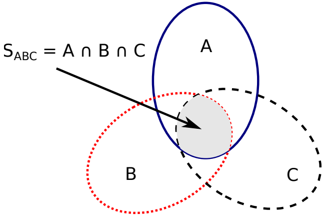
- Comparison of rules based on newly introduced concepts: multi-property set, relationship combination operator "dot r" ($\cdot r$), matchmap and actmap

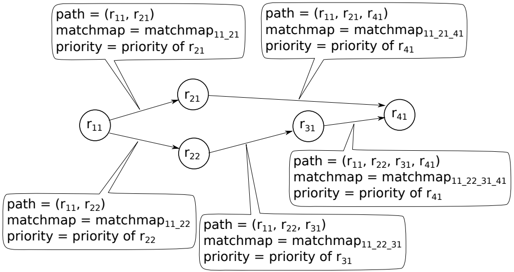


# Overview of the Measures (1)



- Comparison of rules based on newly introduced concepts: multi-property set, relationship combination operator “dot r” ($\cdot r$), matchmap and actmap
- Rule graph
- Input from control applications
- Algorithms



# Overview of the Measures (2)



Local conflict's pattern e.g.,

$$Correlation : priority_i = priority_j, \ \ match_i \subseteq match_j \ \vee \ match_i \supseteq match_j, action_i \neq action_j$$

# Overview of the Measures (2)



Local conflict's pattern e.g.,

$Correlation : priority_i = priority_j,\ match_i \subseteq match_j \ \vee \ match_i \supseteq match_j, action_i \neq action_j$

$\Rightarrow$ **Compare rules** in the same device based on their priority, match, action components

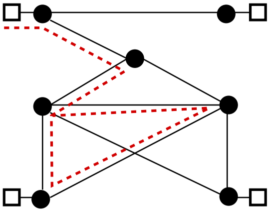
# Overview of the Measures (2)

Local conflict's pattern e.g.,

$Correlation : priority_i = priority_j, \; match_i \subseteq match_j \; \vee \; match_i \supseteq match_j, action_i \neq action_j$

$\Rightarrow$ **Compare rules** in the same device based on their priority, match, action components

Distributed conflict: e.g., downstream traffic loop



# Overview of the Measures (2)



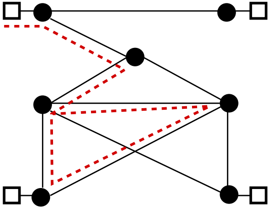
Local conflict's pattern e.g.,

$Correlation : priority_i = priority_j, \; match_i \subseteq match_j \; \vee \; match_i \supseteq match_j, action_i \neq action_j$

$\Rightarrow$ **Compare rules** in the same device based on their priority, match, action components

Distributed conflict: e.g., downstream traffic loop

$\Rightarrow$ Build the **rule graph** based on the connections between rules in different devices



# Overview of the Measures (2)



Local conflict's pattern e.g.,

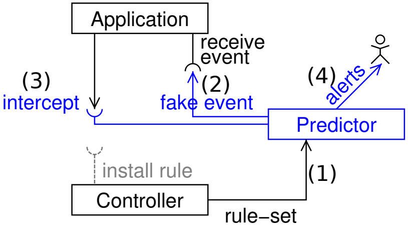
$Correlation : priority_i = priority_j,\ \ match_i \subseteq match_j \ \vee \ match_i \supseteq match_j, action_i \neq action_j$

$\Rightarrow$ **Compare rules** in the same device based on their priority, match, action components

Distributed conflict: e.g., downstream traffic loop

$\Rightarrow$ Build the **rule graph** based on the connections between rules in different devices

Predict hidden conflicts by speculative provocation



# Overview of the Measures (2)



Local conflict's pattern e.g.,

$Correlation : priority_i = priority_j, \;\; match_i \subseteq match_j \;\vee\; match_i \supseteq match_j, action_i \neq action_j$

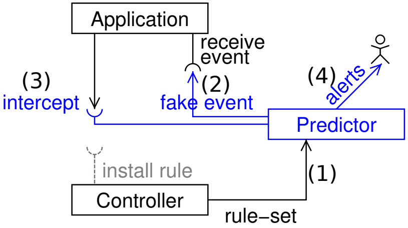
$\Rightarrow$ **Compare rules** in the same device based on their priority, match, action components

Distributed conflict: e.g., downstream traffic loop

$\Rightarrow$ Build the **rule graph** based on the connections between rules in different devices

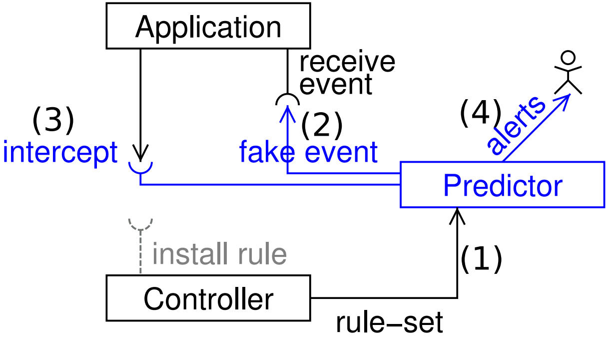
Predict hidden conflicts by speculative provocation

$\Rightarrow$ Detect hidden conflicts with **input from control applications**



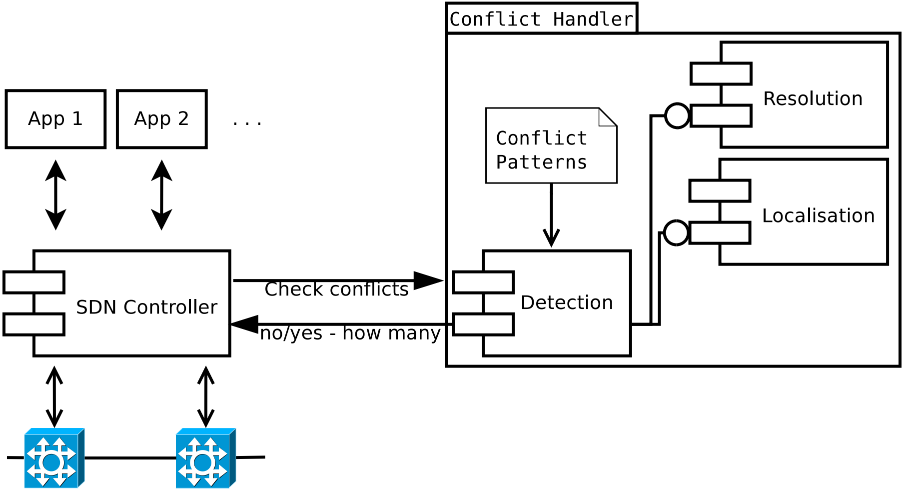
# Hidden Conflicts Predictor





# Hidden Conflicts Predictor in Prototype

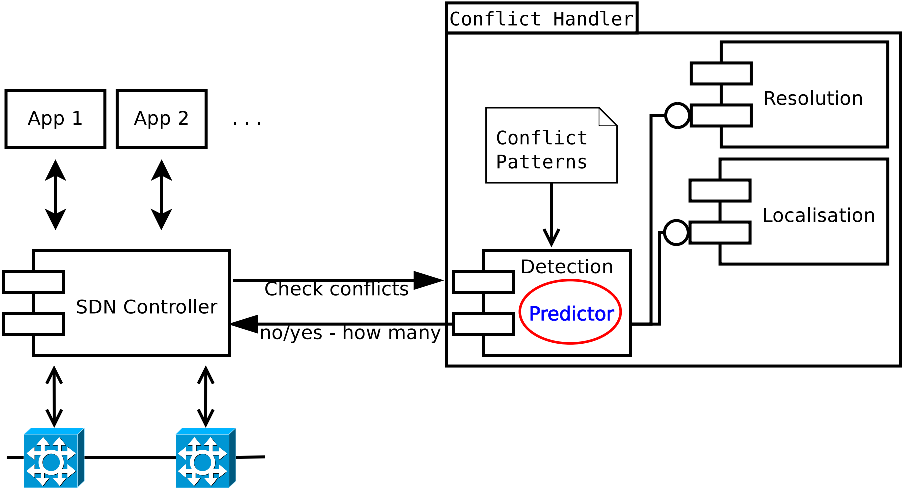





Tran, C.N. and Danciu, V., 2019. A General Approach to Conflict Detection in Software-Defined Networks. SN Computer Science, 1(1), p.9.

# Hidden Conflicts Predictor in Prototype






Tran, C.N. and Danciu, V., 2019. A General Approach to Conflict Detection in Software-Defined Networks. SN Computer Science, 1(1), p.9.

# Comparing SDN rules



Rigidity of the existing solutions [1,2] e.g., match fields of a rule must follow the pattern $< protocol >< src\_ip >< src\_port >< dst\_ip >< dst\_port >$

[1] Al-Shaer, Ehab, Hazem Hamed, Raouf Boutaba and Masum Hasan: *Conflict classification and analysis of distributed firewall policies.* IEEE Journal on Selected Areas in Communications, 23(10):2069–2084, 2005

[2] Pisharody, Sandeep: *Policy Conflict Management in Distributed SDN Environments.* PhD thesis, Arizona State University, 2017

# Comparing SDN rules



Rigidity of the existing solutions [1,2] e.g., match fields of a rule must follow the pattern $< protocol >< src\_ip >< src\_port >< dst\_ip >< dst\_port >$

$\Rightarrow$ **multi-property set**, relationship combination operator **"dot r"** $(\cdot r)$ , **matchmap** and **actmap**

[1] Al-Shaer, Ehab, Hazem Hamed, Raouf Boutaba and Masum Hasan: *Conflict classification and analysis of distributed firewall policies.* IEEE Journal on Selected Areas in Communications, 23(10):2069–2084, 2005

[2] Pisharody, Sandeep: *Policy Conflict Management in Distributed SDN Environments.* PhD thesis, Arizona State University, 2017

# Normalizing SDN Rules by Matchmap, Actmap



The match and action components can be expressed differently in SDN rules, e.g.,:

*rule 1: priority=3, match={ipv4_dst=192.168.2.2, ip_proto=6, tcp_dst=80}, action={output:3}*

*rule 2: priority=2, match={ipv4_src=192.168.1.0/24, ip_proto=6}, action={set_field:ipv4_src=192.168.1.3, output:2}*

# Normalizing SDN Rules by Matchmap, Actmap



The match and action components can be expressed differently in SDN rules, e.g.,:

*rule 1: priority=3, match={ipv4_dst=192.168.2.2, ip_proto=6, tcp_dst=80}, action={output:3}*

*rule 2: priority=2, match={ipv4_src=192.168.1.0/24, ip_proto=6}, action={set_field:ipv4_src=192.168.1.3, output:2}*

*Matchmap* and *actmap* can be employed for normalizing these rules uniformly, so that they can be compared using multi-property set and $\cdot r$:

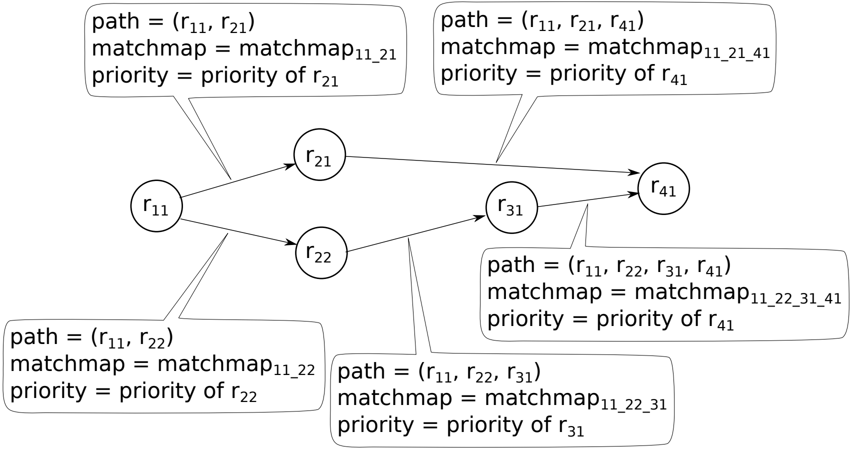
*rule 1: priority=3,* **matchmap**=*{ipv4_src=any, ipv4_dst=192.168.2.2, ip_proto=6, tcp_dst=80},* **actmap**=*{set_field:none, output:3}*

*rule 2: priority=2,* **matchmap**=*{ipv4_src=192.168.1.0/24, ipv4_dst = any, ip_proto=6, tcp_dst = any},* **actmap**=*{set_field:ipv4_src=192.168.1.3, output:2}*

## Rule Graph

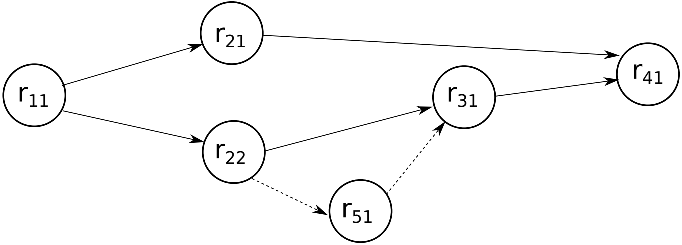


- A directed graph
- A vertex can represent a rule, an end-point, traffic drop or traffic loop



# Rule Graph

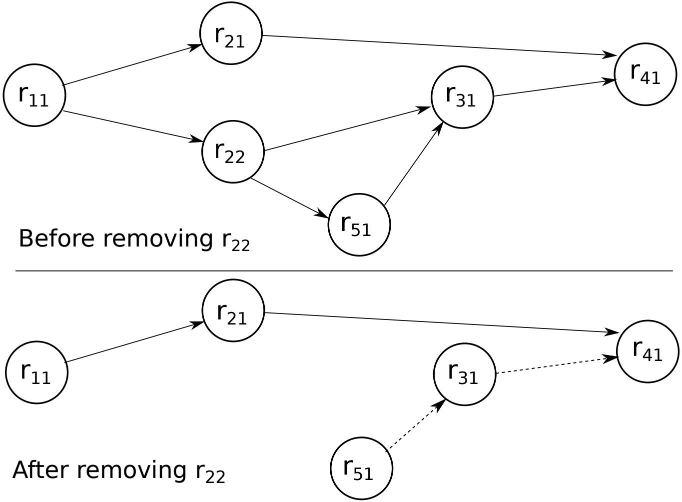
Add a rule to the rule graph



# Rule Graph



Remove a rule from the rule graph



# Input from Control Applications

```
[load_balancer]
  ts = 7 5 6                # target switches
  ip_src = 192.168.1.1      # source IPv4 addresses
  ip_dst = 192.168.2.0/24   # destination IPv4 addresses
  ip_proto = tcp udp        # transport protocols

[traffic_engineering]
  ts = 3 4
  ip_dst = 192.168.2.0/24
  ip_proto = udp
```

# Input from Control Applications

```
[eplb]
ts = 7 5 6 # a list of target switches
ipv4_src = 192.168.1.1 # a list of source IPv4 addresses
ipv4_dst = 192.168.1.3 192.168.1.4 # a list of destination IPv4 addresses
ip_proto = 6 17 # a list of protocols, which includes TCP and UDP in this case

[pplb4d]
ts = 3 4
ipv4_dst = 192.168.1.3, 192.168.2.0/24
ip_proto = 17 # UDP
```

# Detected Results in Designed Cases



For both MWN and Stanford topologies:

| Test | Local conflicts | | | | | Traffic Loop | Traffic Drop | Hidden conflicts ESLH |
|---|---|---|---|---|---|---|---|---|
| | Shadowing | Generalization | Redundancy | Correlation | Overlap | | | |
| 1 | 1/1 | 1/1 | 1/1 | 1/1 | 1/1 | 1/1 | 1/1 | 1/1 |
| 2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 |
| 3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 |
| 4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 |
| 5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 |

Detection of conflicts related to packet modification:

| Test | MWN | Stanford |
|---|---|---|
| 1 | 2/2/2 | 2/1/1 |
| 2 | 5/5/5 | 2/1/1 |
| 3 | 6/6/6 | 4/2/2 |
| 4 | 8/8/8 | 4/2/2 |
| 5 | 10/7/7 | 2/2/2 |

$\Rightarrow$ All conflicts are precisely identified

# Dimensions in Randomly Checking Cases (MWN)



| Dimensions | Test space for MWN test-bed |
|---|---|
| App config. | Each app has 1 config. |
| App start order | Same |
| App priority | All combinations |
| Target switches | EpLB:1, PPLB4S:2, HS:5, PE:10 |
| Ept traf. prof. | CBR |
| Ept combi. (src–>dst) | {3 4 7 8 13 14 15 16 19 21} –>{1 2 5 6 17 18} |
| Topology | MWN |
| Transport type | TCP/UDP |
| # Experiments | 145 ( >62 hours) |

EpLB: End-point Load Balancer, HS: Host Shadowing
PPLB4S: Source-based Passive Path Load Balancer, PE: Path Enforcer
CBR: Constant Bit Rate

# Dimensions in Randomly Checking Cases



| Dimensions | Test space for MWN test-bed | Test space for Stanford test-bed |
|---|---|---|
| App config. | Each app has 1 config. | Each app has 1 config. |
| App start order | Same | Same |
| App priority | All combinations | All combinations |
| Target switches | EpLB:1, PPLB4S:2, HS:5, PE:10 | EpLB: 15 16, PPLB4S: 5 6, PPLB4D: 1 2 |
| Ept traf.prof. | CBR | CBR |
| Ept combi. (src–>dst) | {3 4 7 8 13 14 15 16 19 21} –>{1 2 5 6 17 18} | {9 10 11 12 13 14} –>{1 2 3 4 5 6 7 8} |
| Topology | MWN | Stanford |
| Transport type | TCP/UDP | TCP/UDP |
| # Experiment | 145 ( >62 hours) | 22 (> 9 hours) |

# Detected Results in Randomly Checking Cases



MWN topology:

| Test | App Priority | # rules | Local conflicts | | | | | Traffic Loop | Traffic Drop | HC ESLH |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Sha | Gen | Red | Cor | Ove | | | |
| 1 | (2,2,2,2) | 790 | 0/0/0 | 0/0/0 | 0/0/0 | 27/10/10 | 0/0/0 | 0/0/0 | 0/0/0 | 60/10/10 |
| 2 | (2,2,3,4) | 803 | 0/0/0 | 0/0/0 | 0/0/0 | 26/10/10 | 0/0/0 | 0/0/0 | 0/0/0 | 60/10/10 |
| 3 | (3,2,2,3) | 816 | 0/0/0 | 0/0/0 | 0/0/0 | 27/10/10 | 0/0/0 | 0/0/0 | 0/0/0 | 60/10/10 |
| 4 | (3,5,2,4) | 789 | 0/0/0 | 0/0/0 | 0/0/0 | 25/10/10 | 0/0/0 | 0/0/0 | 0/0/0 | 59/10/10 |
| 5 | (5,4,3,2) | 791 | 0/0/0 | 0/0/0 | 0/0/0 | 24/10/10 | 0/0/0 | 0/0/0 | 0/0/0 | 60/10/10 |

Stanford topology:

| Test | App Priority | # rules | Local conflicts | | | | | Traffic Loop | Traffic Drop | HC ESLH |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Sha | Gen | Red | Cor | Ove | | | |
| 1 | (2,2,2) | 650 | 0/0/0 | 0/0/0 | 0/0/0 | 4/4/4 | 0/0/0 | 0/0/0 | 0/0/0 | 34/10/10 |
| 2 | (2,3,4) | 672 | 0/0/0 | 0/0/0 | 0/0/0 | 5/5/5 | 0/0/0 | 0/0/0 | 0/0/0 | 34/10/10 |
| 3 | (3,2,2) | 670 | 0/0/0 | 0/0/0 | 0/0/0 | 5/5/5 | 0/0/0 | 0/0/0 | 0/0/0 | 35/10/10 |
| 4 | (3,4,2) | 662 | 0/0/0 | 0/0/0 | 0/0/0 | 5/5/5 | 0/0/0 | 0/0/0 | 0/0/0 | 35/10/10 |
| 5 | (4,3,2) | 659 | 0/0/0 | 0/0/0 | 0/0/0 | 5/5/5 | 0/0/0 | 0/0/0 | 0/0/0 | 34/10/10 |

$\Rightarrow$ All randomly checking conflicts are correct

## Conclusions



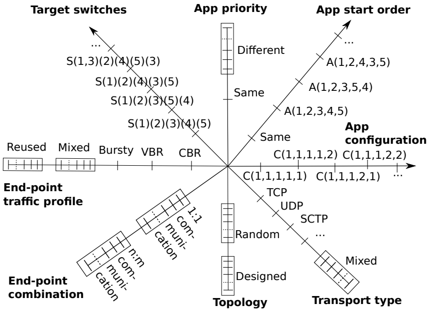
- Experimental approach for researching conflicts: parameter space, methodology
- A framework for automating experiments: more than 11,700 experiments have been conducted
- Conflict classication: 19 conflict classes, hidden conflict are completely new
- Conflict detection with multi-property set, relationship combination operator $\cdot r$, matchmap, actmap, rule graph
- Conflict detection prototype and evaluation: the quality of soundness and completeness is confirmed

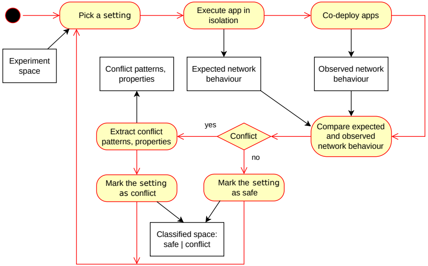
# Conclusions

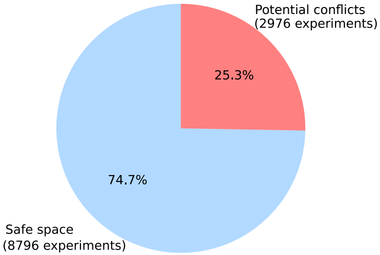


## Experimental approach for researching conflicts

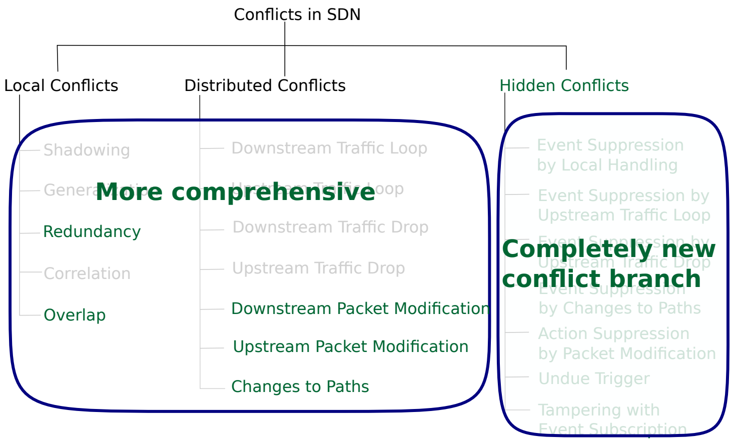
## A framework for automating experiments



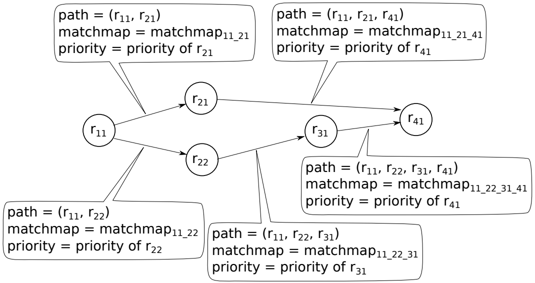



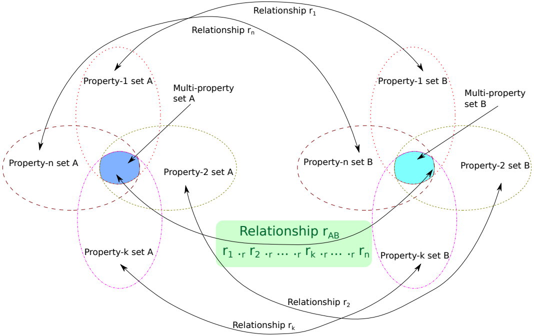



## Conflict classication



# Conflict detection with multi-property set, relationship combination operator $\cdot r$, rule graph



## Conflict detection prototype and evaluation