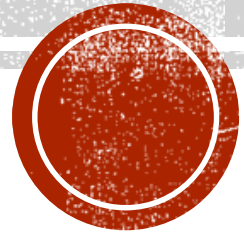
SOFTWARE DEFINED VIRTUAL NETWORKING SECURITY

CHAPTER 10 SECURITY POLICY MANAGEMENT IN DISTRIBUTED SDN ENVIRONMENTS

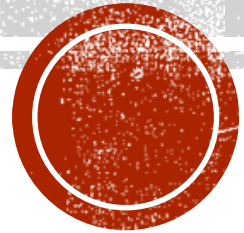


Dijiang Huang, Ankur Chowdhary, and Sandeep Pisharody

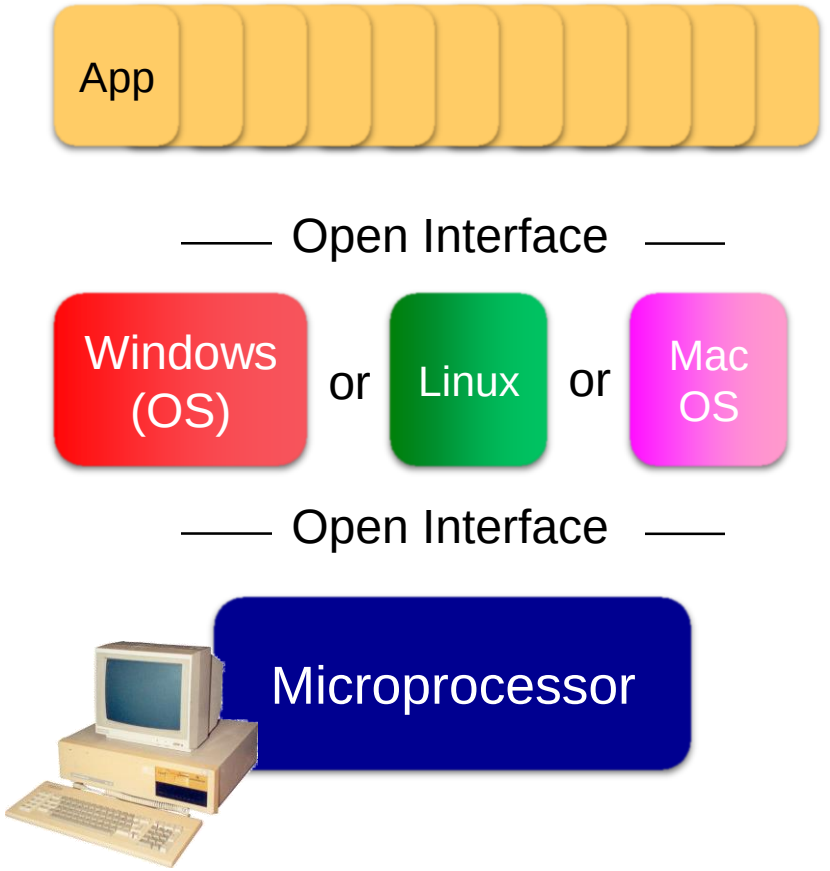
OUTLINE

- SDN paradigm
- Security challenges
  - Flow rule conflicts
- Controller decentralization
- *Brew* Framework
- Future work

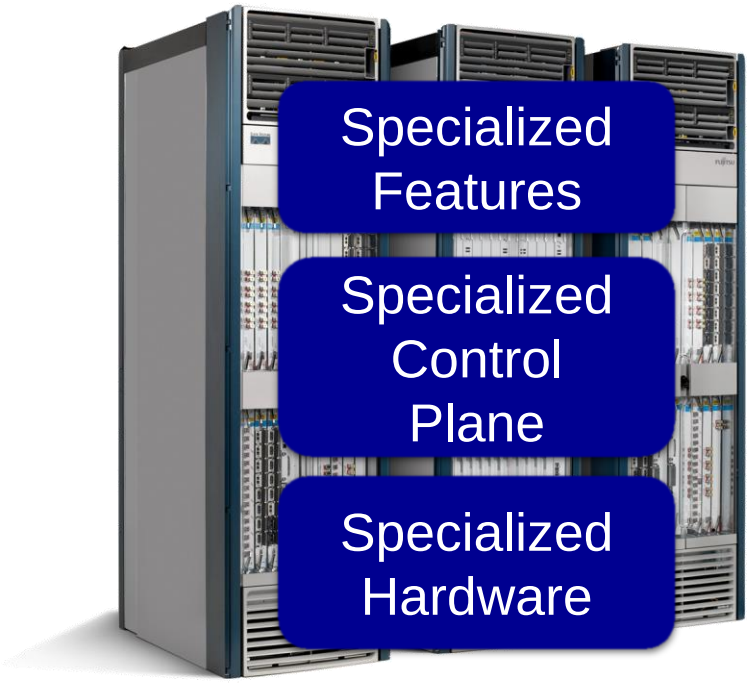
SDN PARADIGM

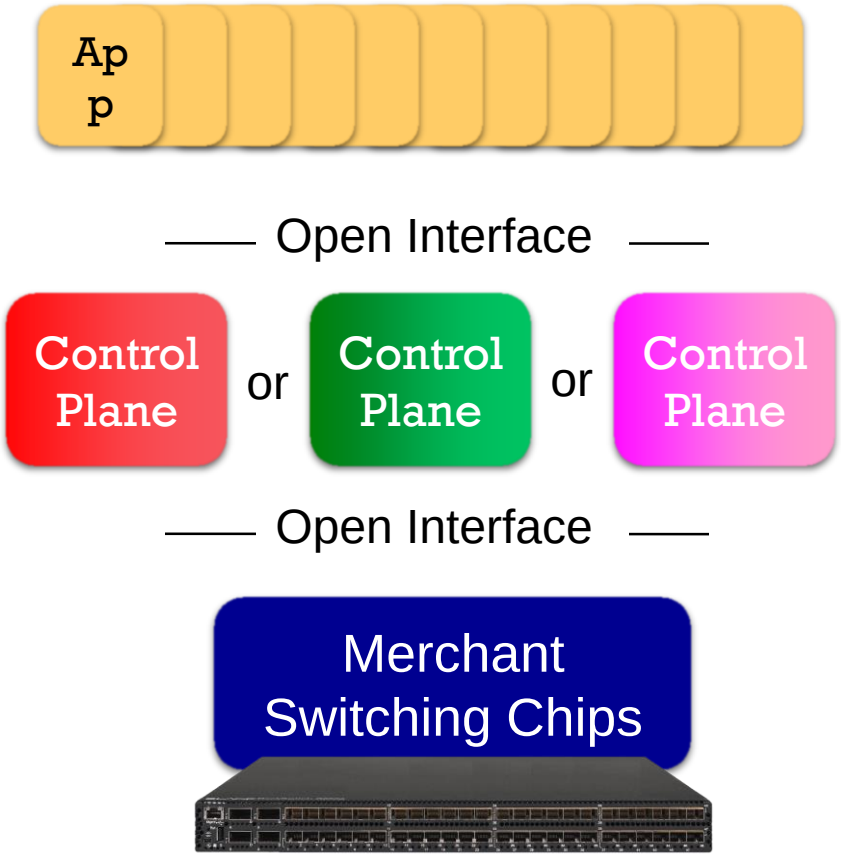


MAINFRAMES TO PC



NETWORK DEVICES TO SDN





SOFTWARE DEFINED NETWORKS

- Separation of control and data plane
- Centralization of control plane
- Programmability of control plane
- Standard and Open API

SDN DATA PLANE

- Simplicity is they name
  - Pattern: match packet header bits
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

SDN CONTROL PLANE

- Brains of the operation
  - Global view of the network
  - Determines what to do with a packet

- Switch state is a function of the global network state
  - All switches may not have the same info

SDN CONTROLLER

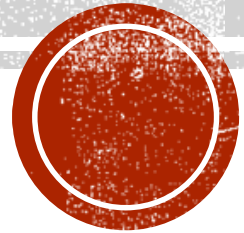
- Single point of failure
- Limits network growth
- Performance bottleneck
  - Scalability
  - Efficiency (switch-controller latency)
  - High latency between controller and some switches

SDN CHALLENGES

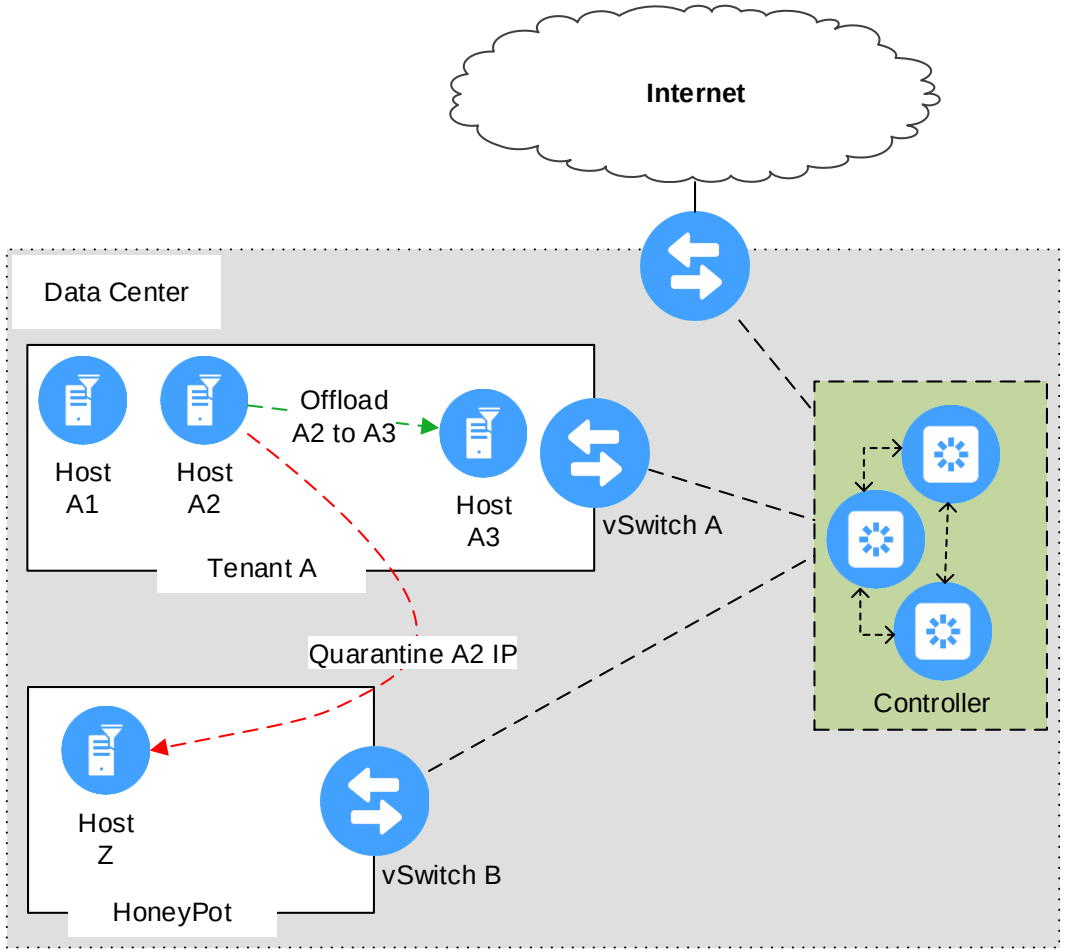
- Since security is centralized, the controller becomes an attractive attack target

- Controller only sees traffic that the switches don't know how to handle
  - Switches only ***think*** they know how to handle it

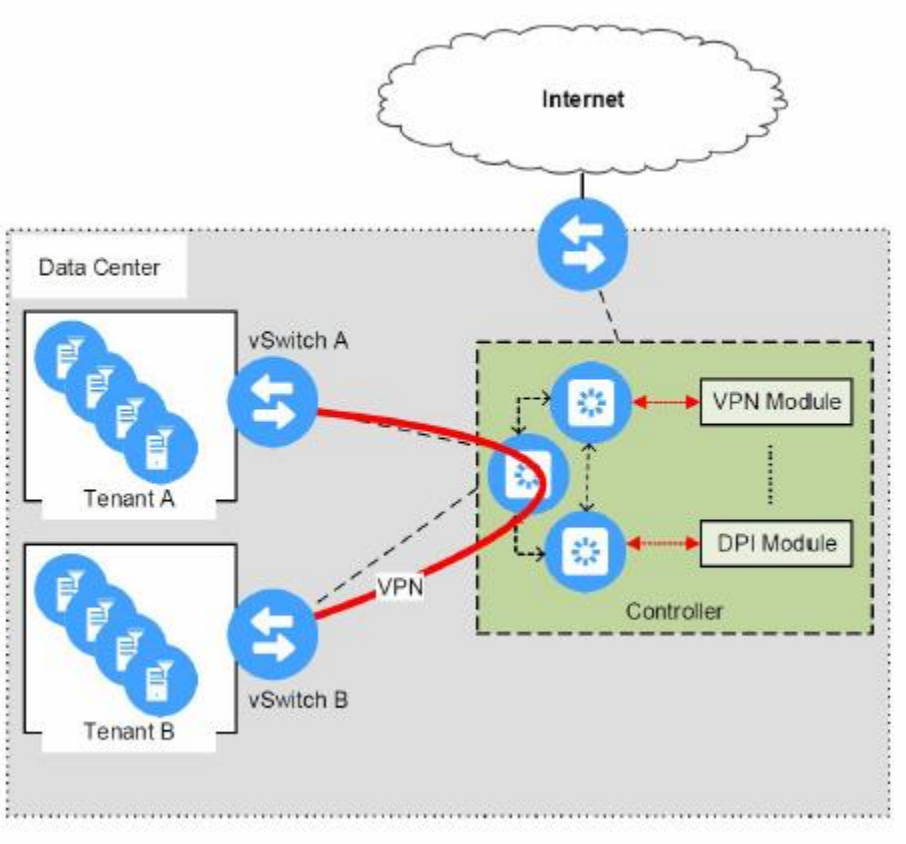
SECURITY CHALLENGES



MOTIVATING EXAMPLE



MOTIVATING EXAMPLE



THREAT MODEL

- Internet threat model
  - Trust thy neighbor
  - System centric

- Asset centric model
  - Protect the Cyber Key Terrain (CKT)

MOTIVATION

- Evolution of threat model

- Defense in depth strategy
  - Security control
  - Audit, reviews
  - Security response plan

MOTIVATION

- Pervasiveness of Internet
- Evolution of threat model
- Defense in depth strategy
  - Security control
  - Audit, reviews
  - Security response plan

FIREWALL EVOLUTION

- Routers
- Security boxes (IPTables)
- Distributed firewalls
- Middleboxes
  - Add SDN!

SECURE & RESILIENT NETWORKING

- Programmable Network MTD
  - Network based countermeasures for Moving Target Defense
    - Requires framework that supports dynamic changes
      - Address changes
      - Topology changes

SECURE & RESILIENT NETWORKING

- Detect vulnerabilities
  - Minimize attack graph
- Detect compromise
- Traffic engineering
  - To minimize exposed attack surface
  - Optimize user experience

SDN FOR SECURITY

- Moving Target Defense
  - Dynamically change network upon detection of an attack
    - Set-Field actions mean you can use L2, L3 and L4 to help move the target
  - Insider attack
    - Detect compromised host
    - Host can be quarantined instantaneously

SDN FOR SECURITY

- Moving Target Defense
  - Honeypots
  - IP address mutation scheme to mask real IP addresses*

- Countermeasure actions can include
  - Reconfigure network
  - Spawn/kill servers
  - Traffic engineering

SECURE & RESILIENT NETWORKING

- Ensure policy compliance
  - Detect conflicts
- Ensure reliability of software modules
  - Reliability metric?

ENHANCING SECURITY IN SDN

- Add diversity to systems
  - Avoiding shared vulnerabilities
- In-built trust between controllers, devices and admin stations
- Self-identify network anomalies and dynamically update configuration

SDN SECURITY MODEL

- Security cannot be enforced by physical topology
- Flow Rules control when or if traffic goes through Security Device (or application)
  - Match/Action combination can be used to specify traditional firewall/IPS/IDS rules
    - Lack of state information

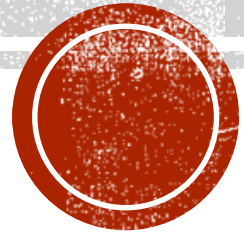
FLOW RULE MANAGEMENT

- Multiple policy generation points
  - Policy based
  - Manual addition
- Interactions between concurrent modules and dynamic network updates
- Priority is not unique

FLOW RULE MANAGEMENT

- Rules have more fields in match conditions and possible actions
- More layers to consider
  - Set-Field actions can dynamically change several headers
  - Cross-layer interaction
- Wildcard matching on rules
  - Networks and subnets have no meaning

SDN FLOW RULE CONFLICTS



FLOW RULE MATCH

- Match fields are expanded as well
  - Ingress Port
  - Ethernet source / destination address
  - Ethernet type
  - IPv4 or IPv6 protocol number
  - IPv4 source / destination address
  - IPv6 source / destination address
  - TCP/UDP source / destination port

FLOW RULE ACTIONS

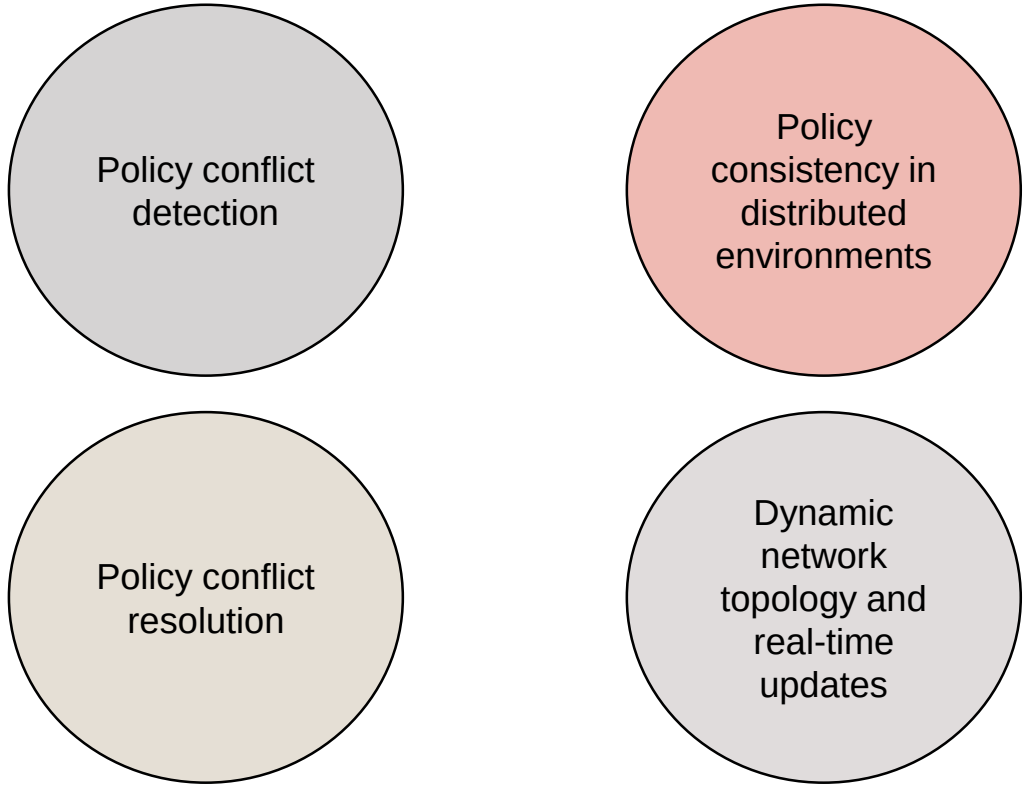
- Action set in traditional firewall
  - Forward (with/without modifying)
  - Drop

- Flow rule actions [OpenFlow 1.3.1]
  - Forward
  - Drop
  - Modify
  - Set-queue
  - Push / Pop tag
  - Set-Field
  - Change TTL

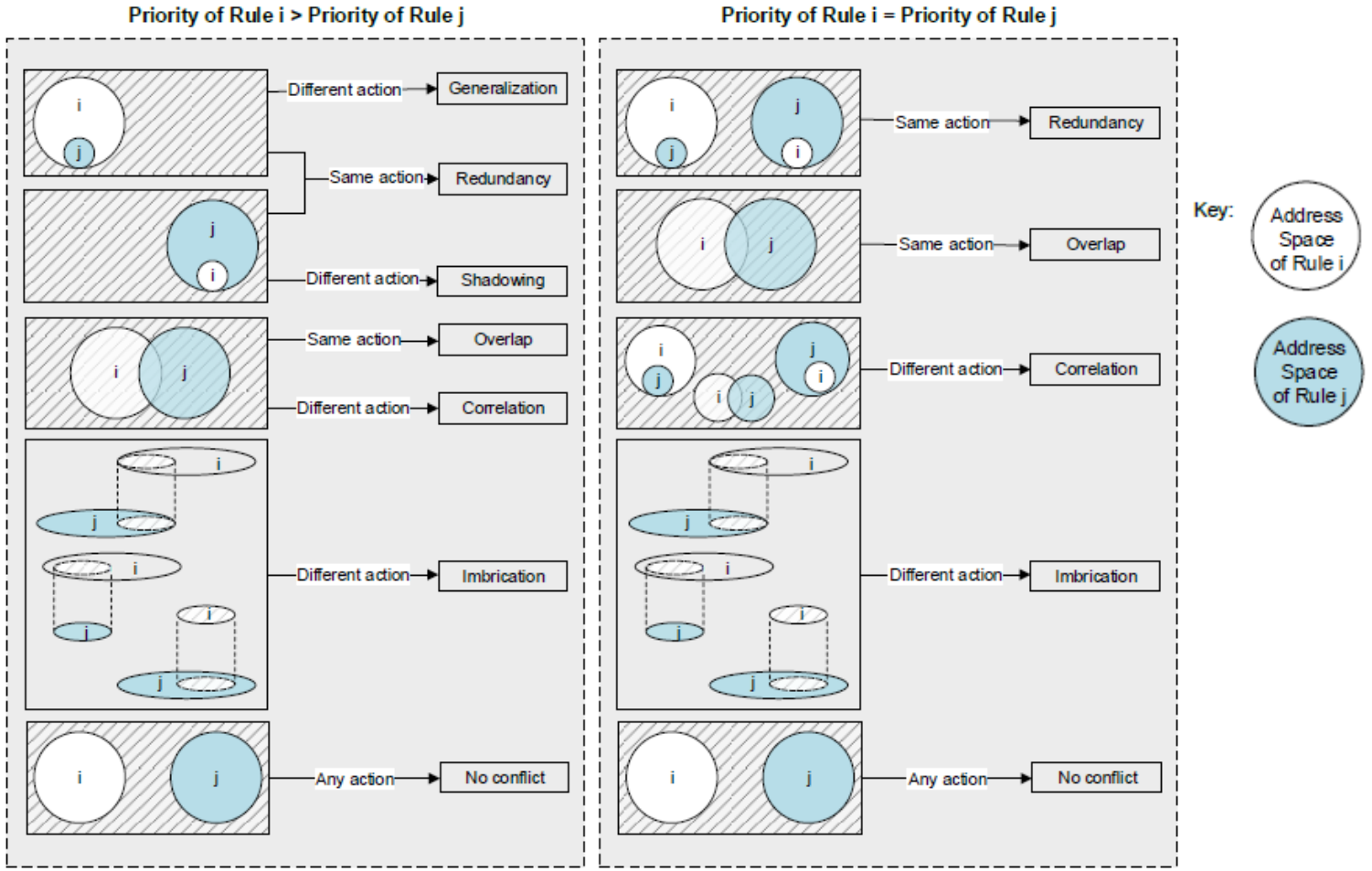
FLOW RULE CHALLENGES

- Rules have more fields in match conditions
- Flow rules include wildcard in rules
- Cross-layer interaction in SDN
  - Set-Field actions can dynamically change several headers
- Interactions between concurrent modules and dynamic network updates
- Priority is not unique

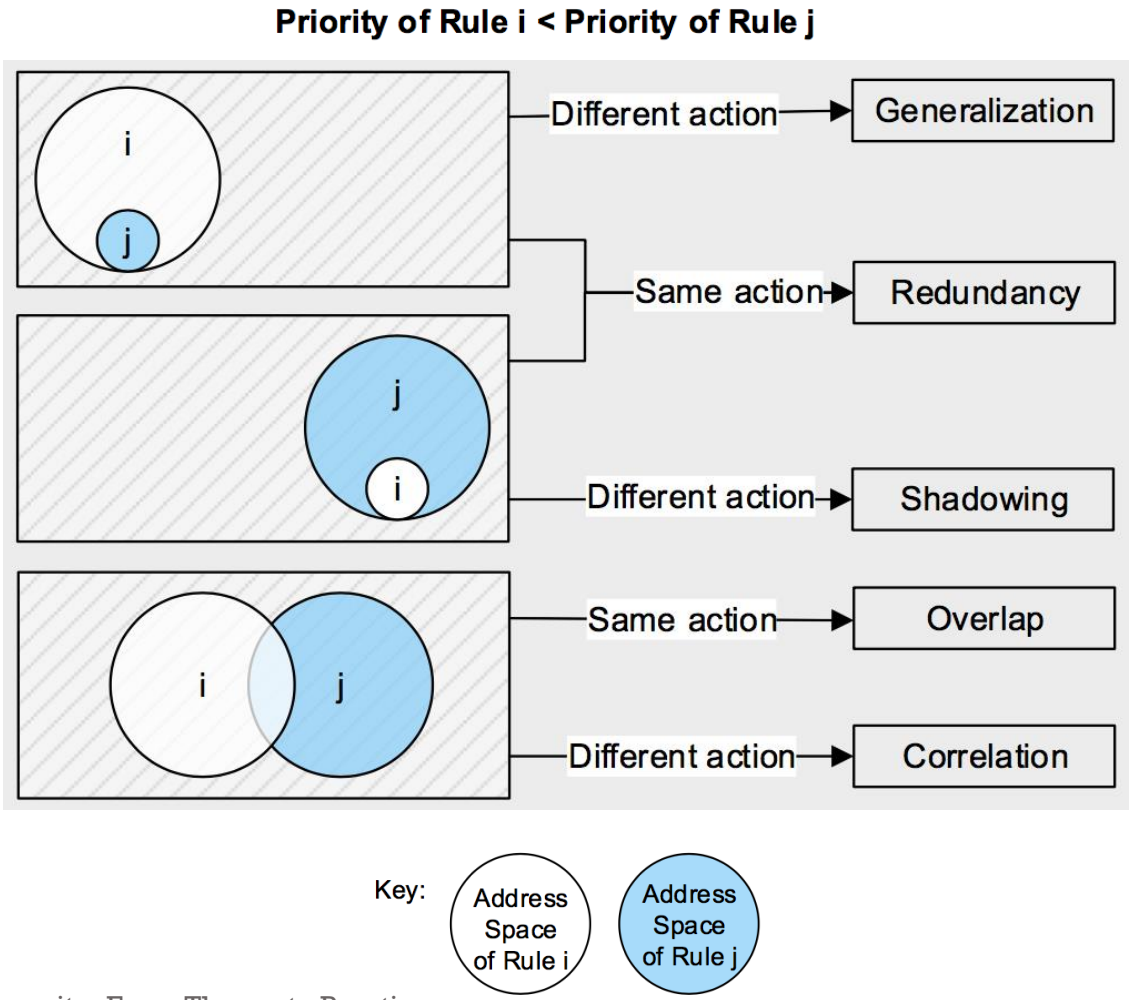
PROBLEM SPACE



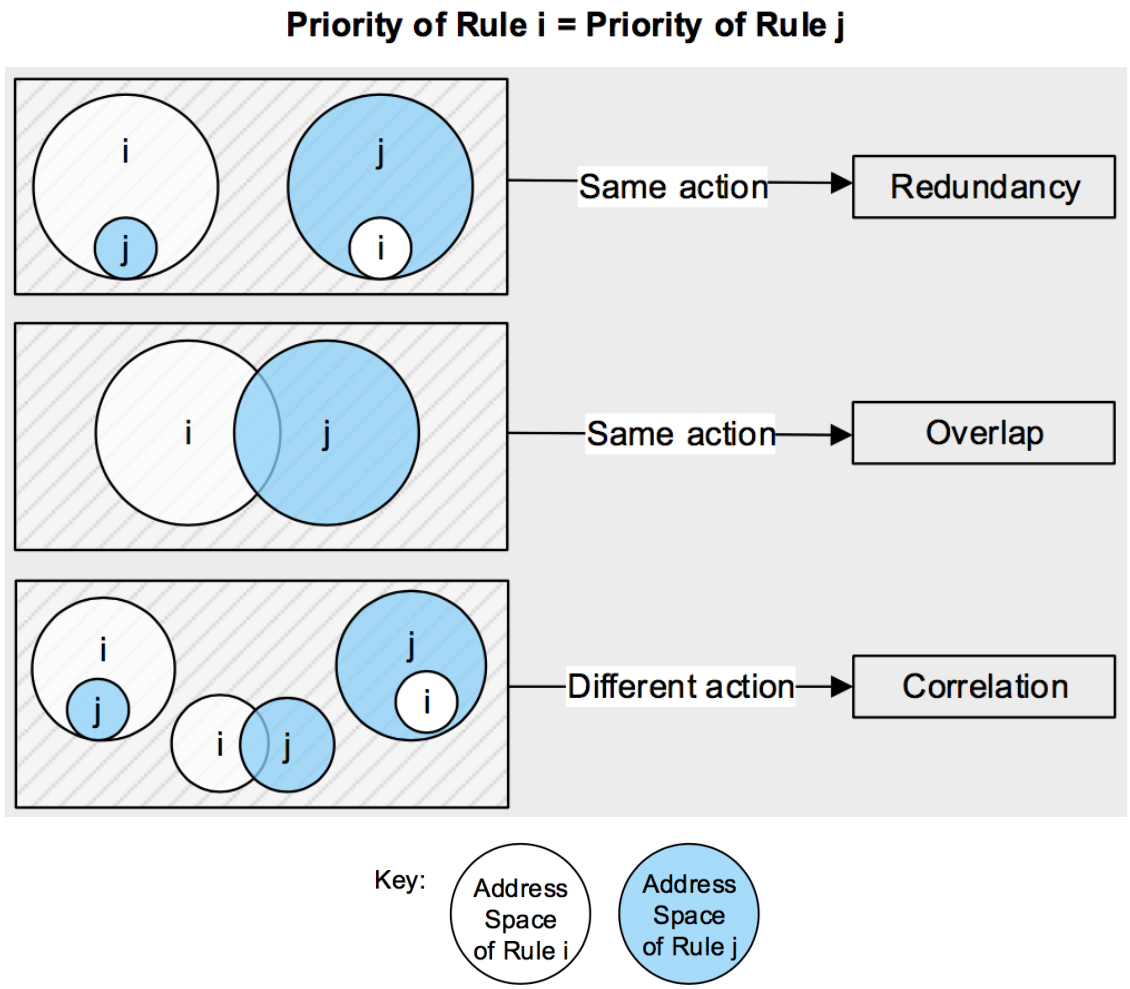
CONFLICT CLASSIFICATION



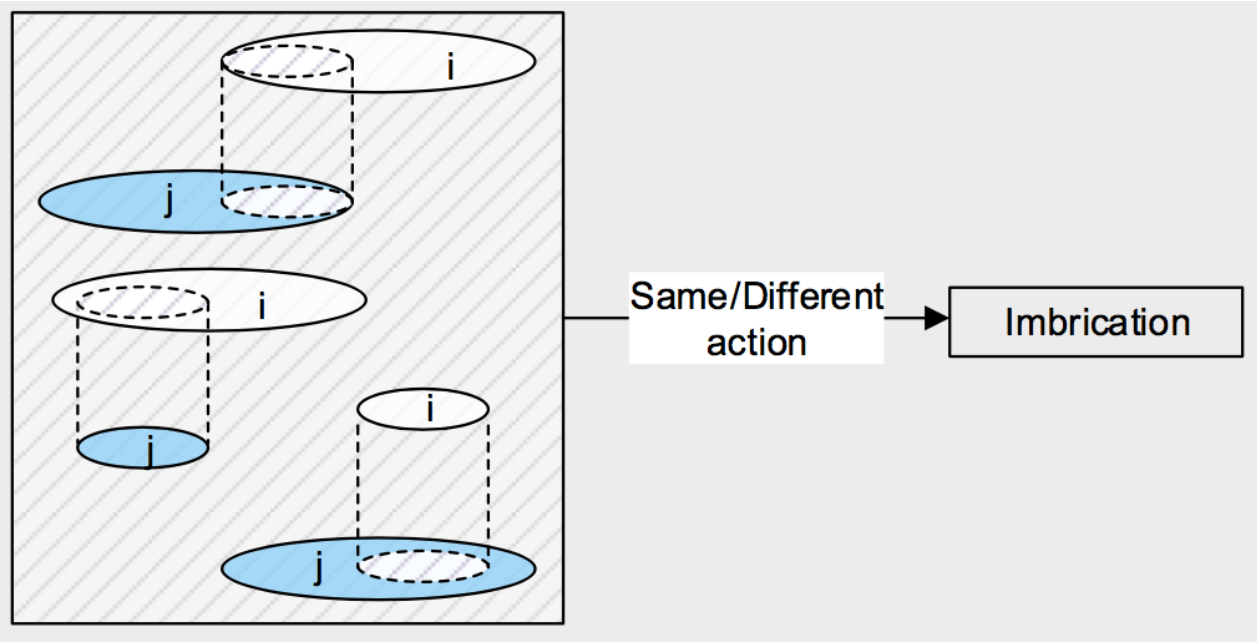
CONFLICT CLASSIFICATION



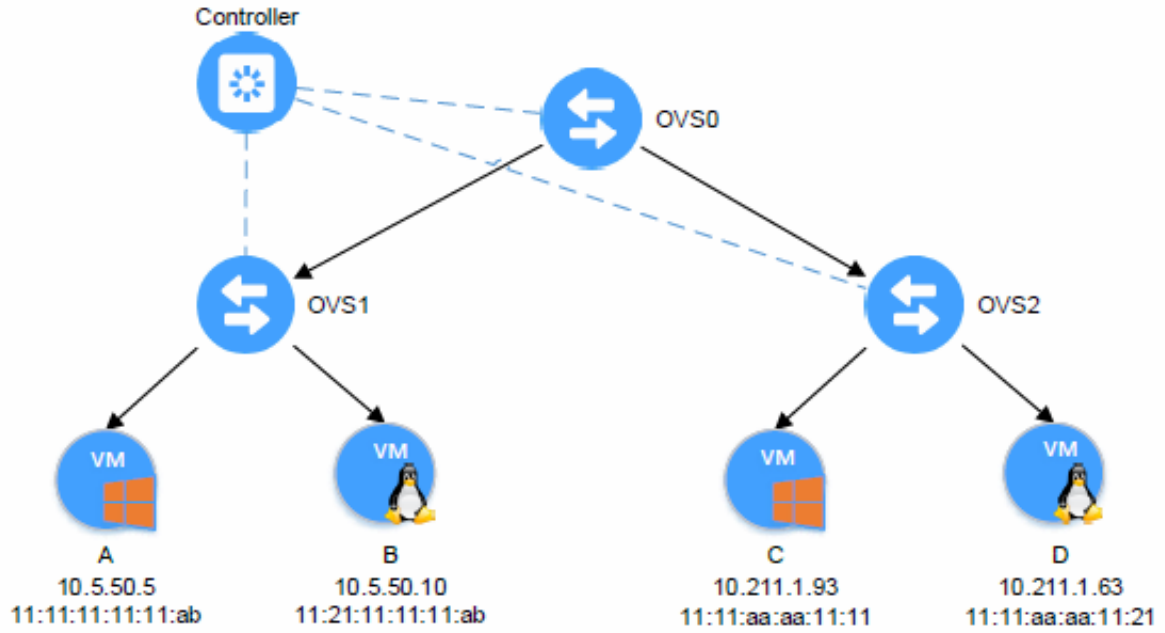
CONFLICT CLASSIFICATION



CONFLICT CLASSIFICATION



CONFLICT CLASSIFICATION




```
cookie==0x2b0000000000000b, duration=926.421s, table=0, n_packets
   =1378, n_bytes=271308, idle_age=77, priority=100 dl_type=0x800
   dl_src=11:11:11:11:11:ab nw_dst=10.211.1.63 actions=NORMAL

cookie=0x2b0000000000003a, duration=949.733s, table=0, n_packets
   =622, n_bytes=957, idle_age=144, priority=100, dl_type=0x800
   dl_src=11:11:11:11:11:ab dl_dst=11:11:aa:aa:11:21 actions=drop
```

MANAGEMENT PROBLEMS

Conflict Detection Problem: The conflict detection problem [6] seeks to find rules $r_i = (p_i, n_i, a_i)$, $r_j = (p_j, n_j, a_j)$ such that $r_i, r_j \in R$ and $n_i = n_j \wedge (a_i \neq a_j \vee p_i \neq p_j)$.

MANAGEMENT PROBLEMS

Packet Classification Problem: For an incoming packet Π_{in} with the network 5-tuple n_{in}, the packet classification problem [6] in firewalls seeks to find out the set $R_m \subseteq R$ where $R_m = \{r_i = (p_i, n_i, a_i) \mid r_i \in R \ \wedge \ n_i = n_{in}\}$. The problem can be further extended to determine rule $r_x = (p_x, n_x, a_x) \in R_m$ such that $p_x > p_y \ \forall \ r_y \in R_m$.

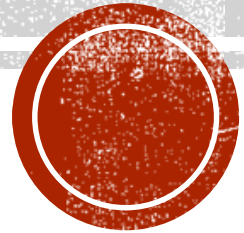
MANAGEMENT PROBLEMS

Conflict Detection Problem: The conflict detection problem [6] seeks to find rules $r_i = (p_i, n_i, a_i)$, $r_j = (p_j, n_j, a_j)$ such that $r_i, r_j \in R$ and $n_i = n_j \wedge (a_i \neq a_j \vee p_i \neq p_j)$.

FLOW RULE CONFLICTS

- More complex than traditional firewalls
  - More layers to consider
  - Set-field actions
  - Wildcard matching on rules
    - Networks and subnets have no meaning
      - 192.202.5.50/24 and 192.168.5.1/24 both match 192.*.5.1/24
  - Cross-layer interaction

CONTROLLER DECENTRALIZATION

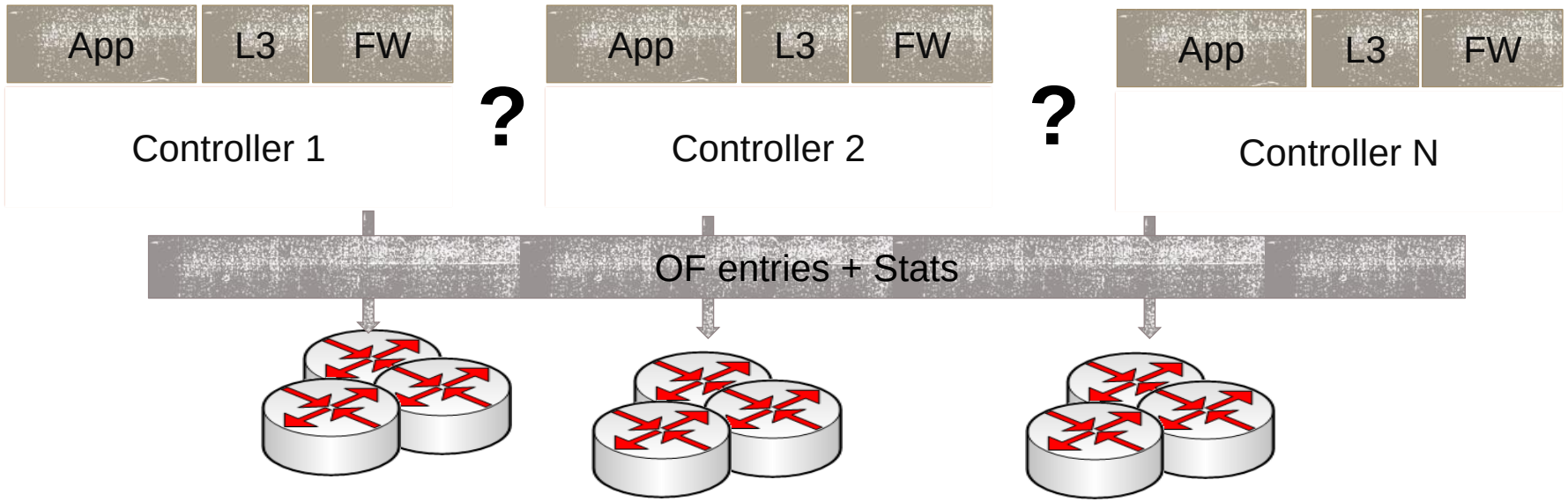


MULTIPLE CONTROLLERS

- Controller placement considerations
  - Reliability
  - Fault tolerance
  - Controller distribution
    - Mesh vs Hierarchy
  - Scalability
- Synchronization/concurrency issues

MULTIPLE CONTROLLERS

- How do you deal with synchronization/concurrency problems

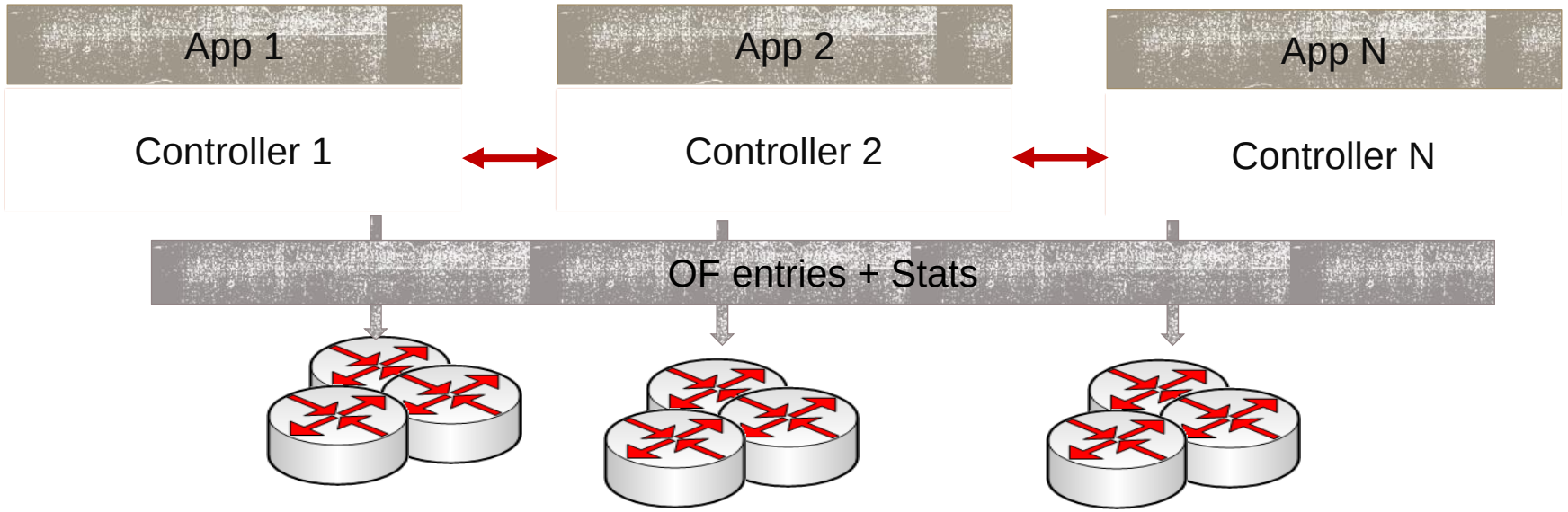


MULTIPLE CONTROLLERS

- Dealing with synchronization/concurrency problems
  - Medium sized
    - Controller can process all events, run all applications
    - Has ability to produce same output
  - Large scale
    - Each controller can't run all the applications or handle all events
      - Need to partition the application/devices

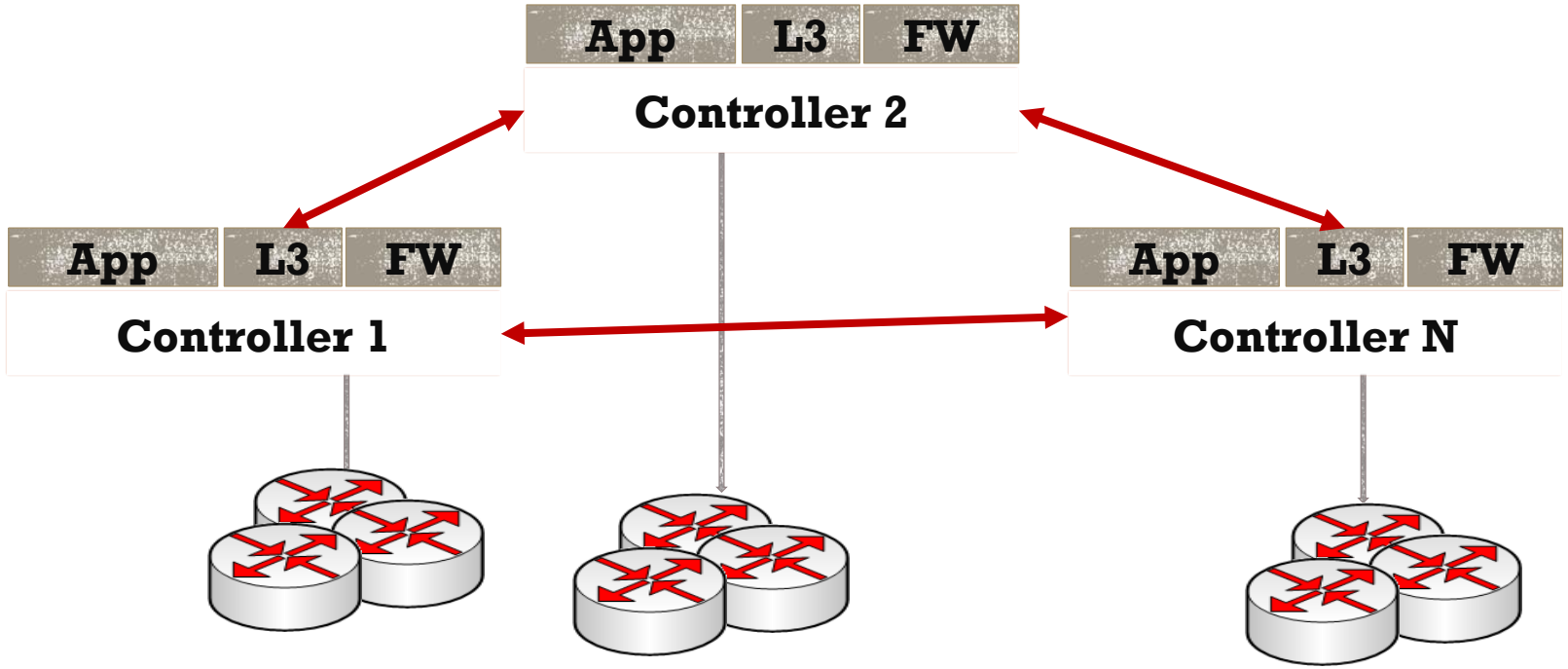
MULTIPLE CONTROLLERS

- Approach 1 - Each controller runs a specific application



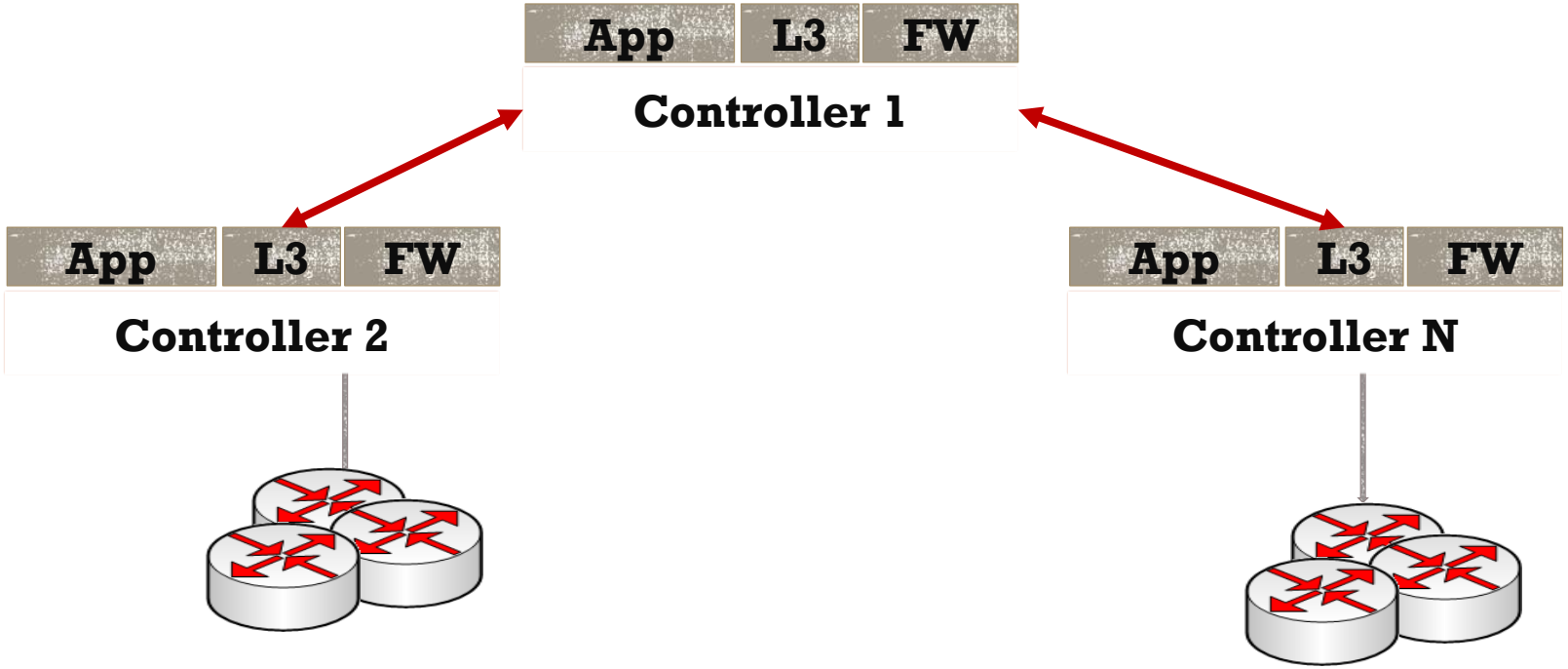
MULTIPLE CONTROLLERS

- Approach 2: All controllers run the same application but for a subset of devices



MULTIPLE CONTROLLERS

- Approach 3: Split application into local, and global



MULTIPLE CONTROLLERS

- Priority/Hierarchy of controllers is important
  - OF switches by default can have multiple controllers
    - Only take responses from "Primary" controller

GLOBAL PRIORITIES

- Host Partitioning
  - Local rules are preferred
- Hierarchical controllers
  - Local (or leaf) controllers have the lowest priority
- Application Partitioning
  - Weighted priorities depending on generating application
    - Security
    - Reliability
    - Manual ordering

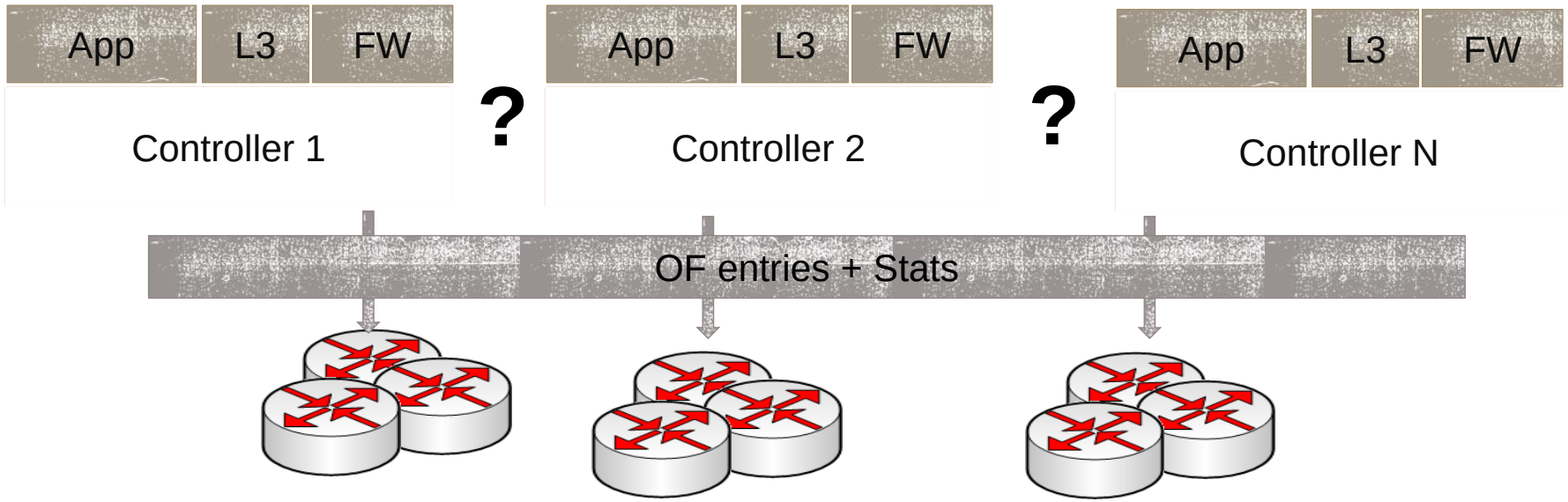
DECENTRALIZED CONTROLLER

- Improved scalability and reliability

- Challenges
  - Synchronization/concurrency issues
  - Partitioning of applications/functionalities
    - Who controls which switch?
    - Who reacts to which events?

CONTROLLER SCALABILITY

- How do applications deal with synchronization/concurrency problems



CONTROLLER SCALABILITY

- How do applications deal with synchronization/concurrency problems
  - Depends on network size

CONTROLLER SCALABILITY

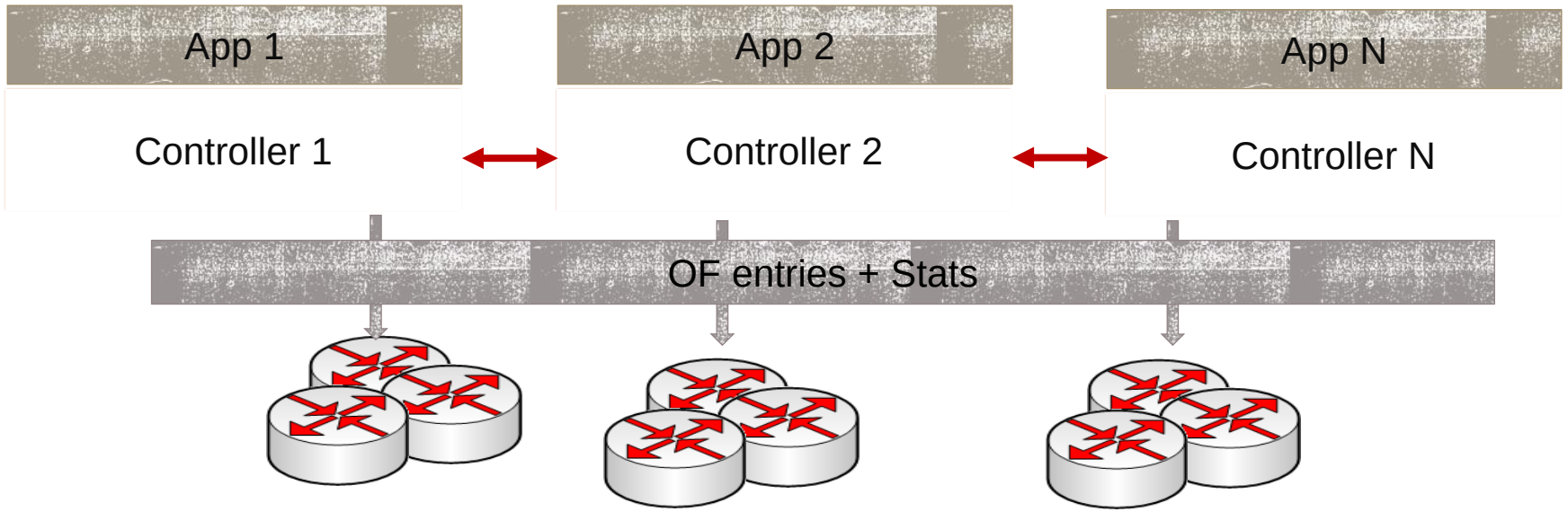
- How do applications deal with synchronization/concurrency problems
  - Medium size
    - Controller can process all events, run all applications
      - Has ability to produce same output

CONTROLLER SCALABILITY

- How do applications deal with synchronization/concurrency problems
  - Large size
    - Each controller can't run all the applications or handle all events
      - Need to partition the application

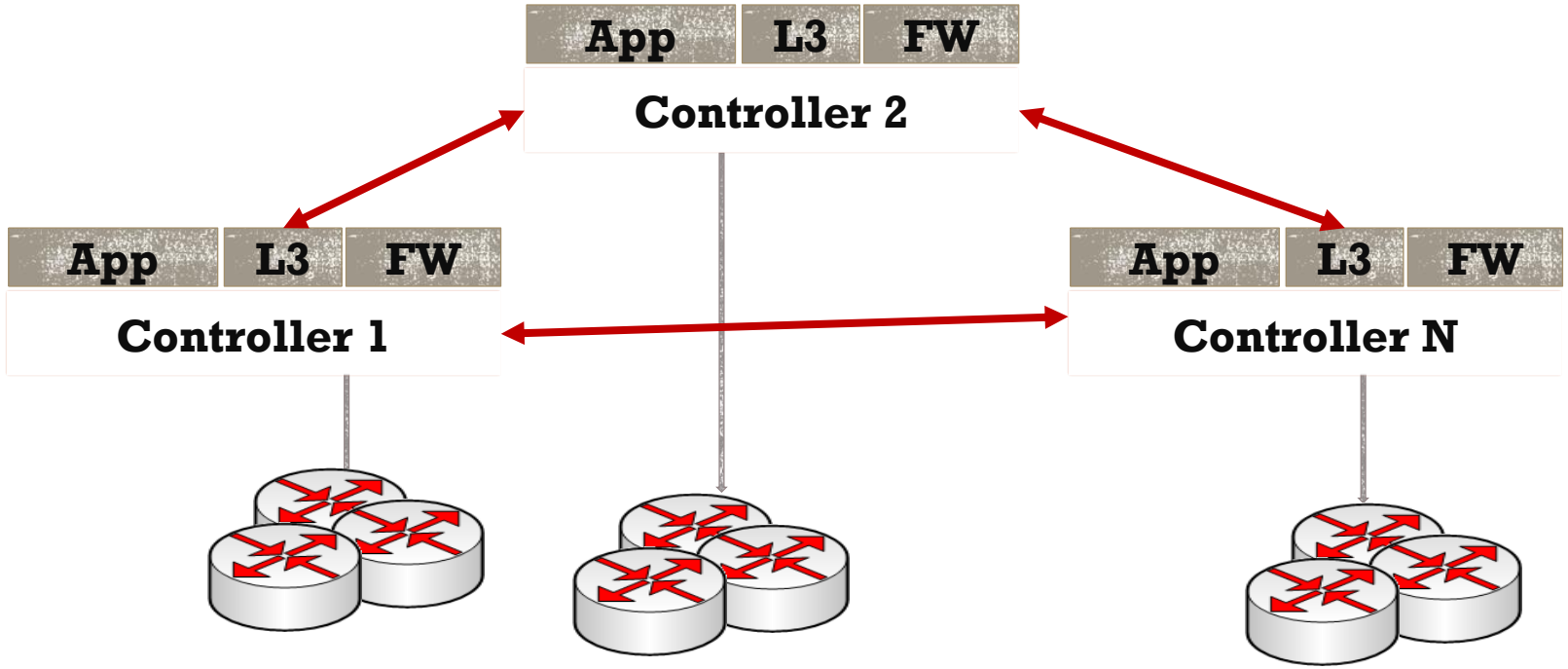
APPLICATION PARTITIONING

- Approach 1 - Each controller runs a specific application



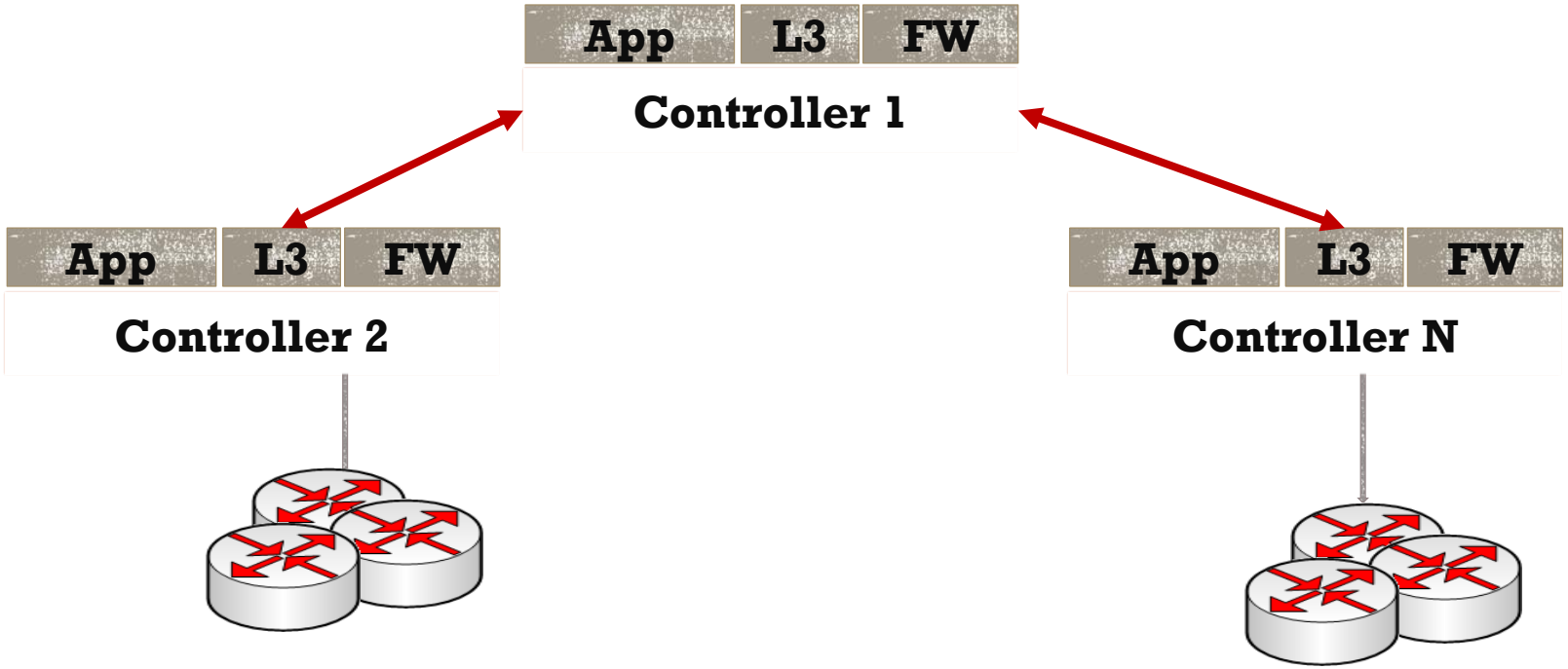
APPLICATION PARTITIONING

- Approach 2: All controllers run the same application but for a subset of devices

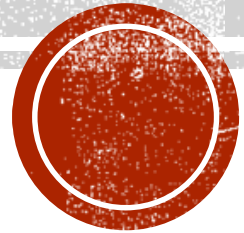


APPLICATION PARTITIONING

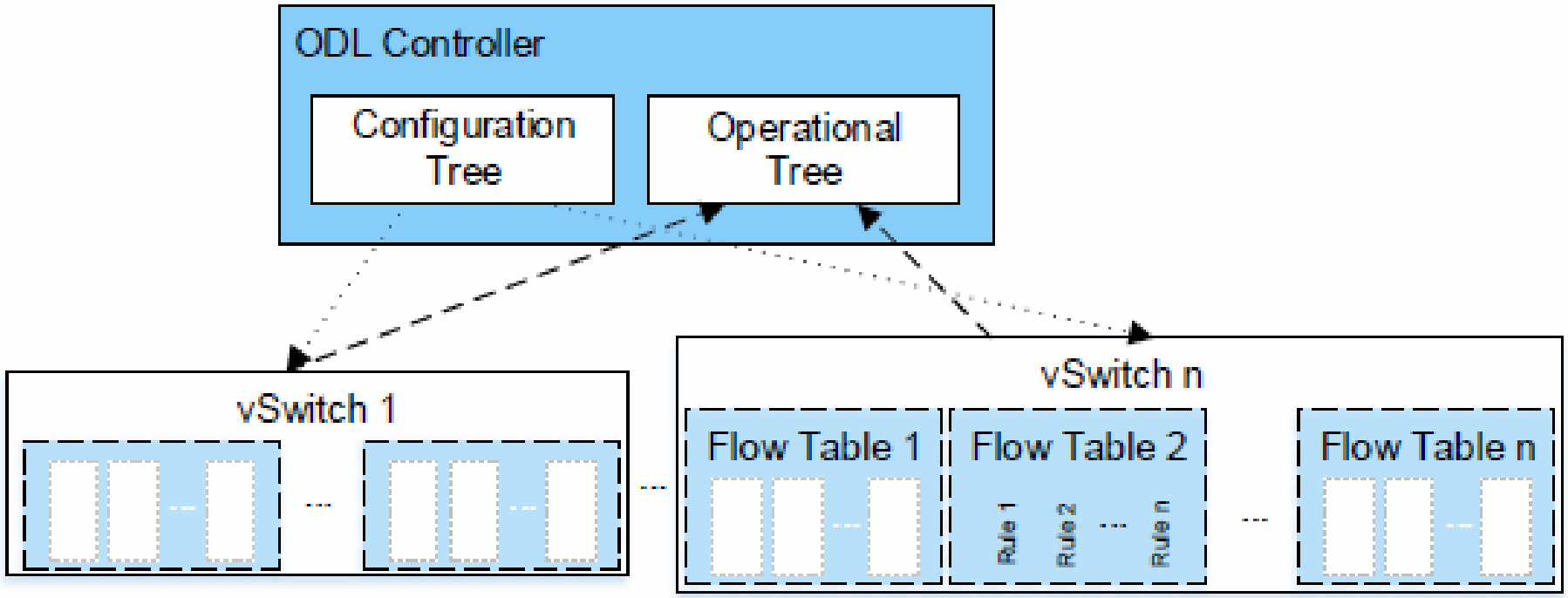
- Approach 3: Split application into local, and global



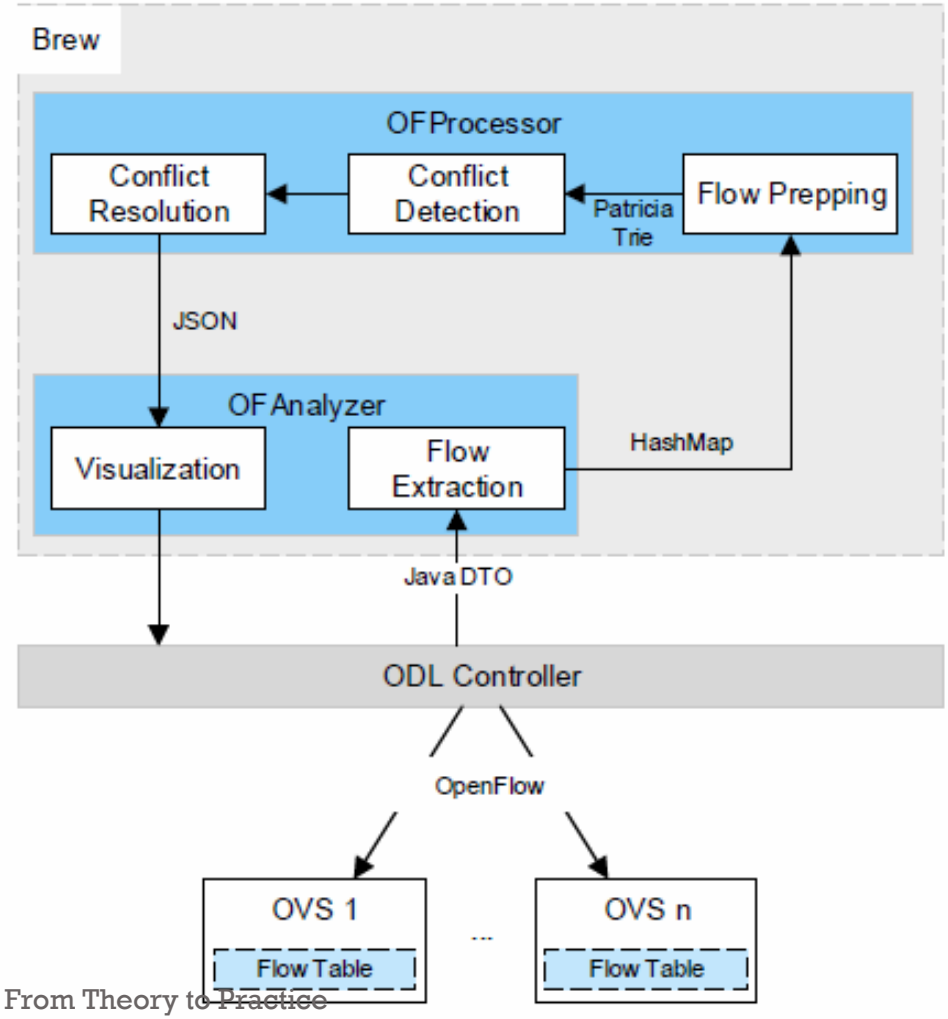
BREW FRAMEWORK



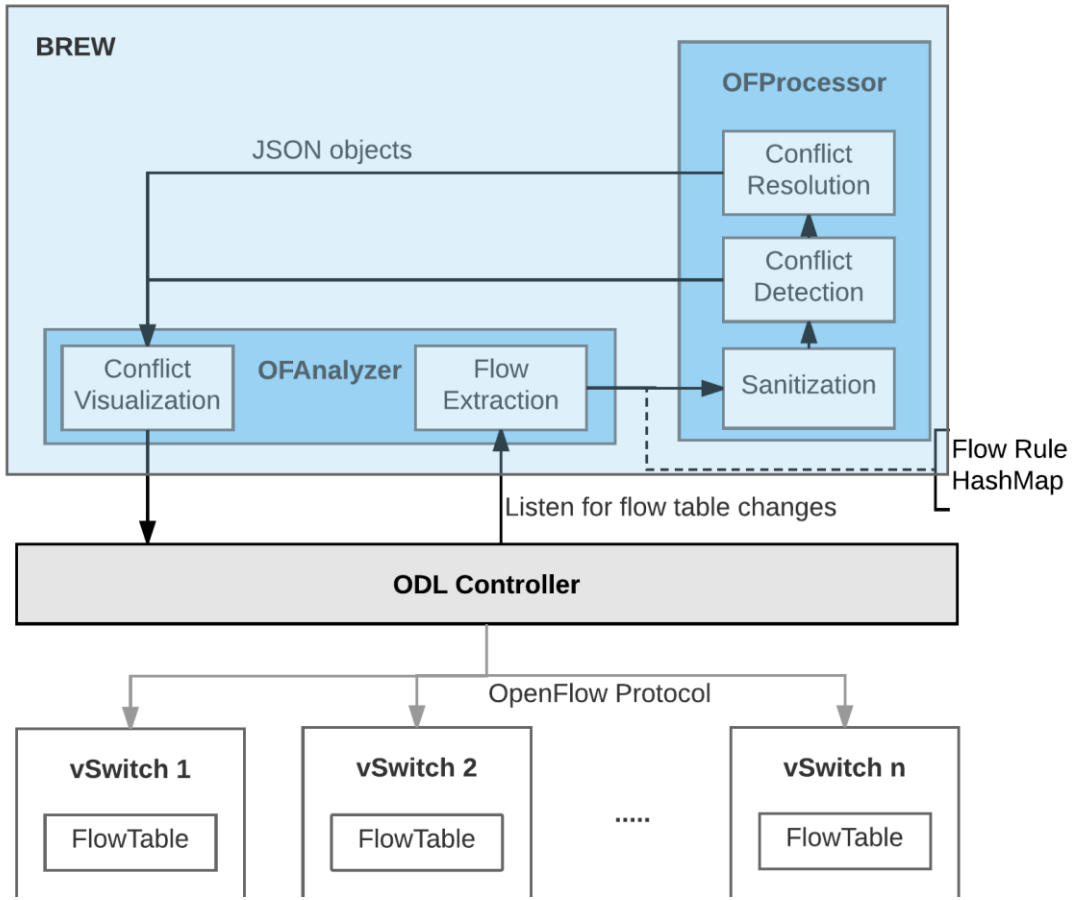
OPENDAYLIGHT



SYSTEM DESIGN

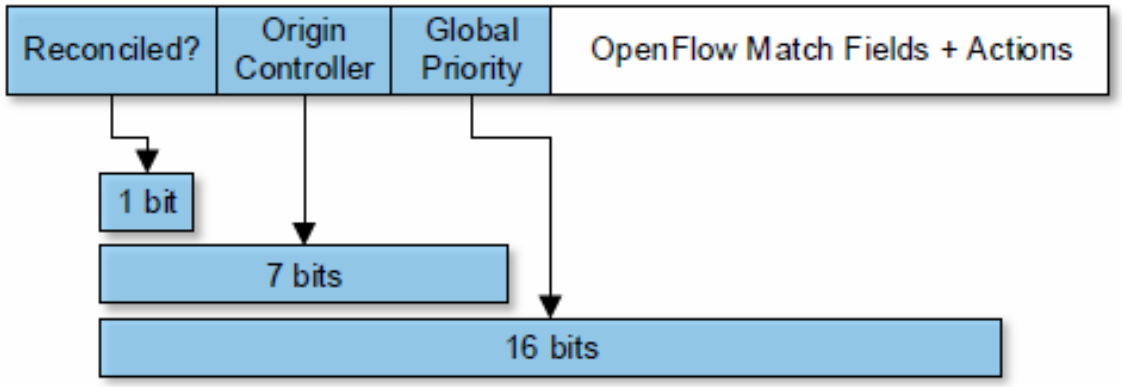


SYSTEM DESIGN



FLOW RULE EXTRACTION

- Intercepts candidate flow rule(s) that is to be added to the flow table
- Concomitant data structure added
  - Global priority assignment



FLOW PREPPING

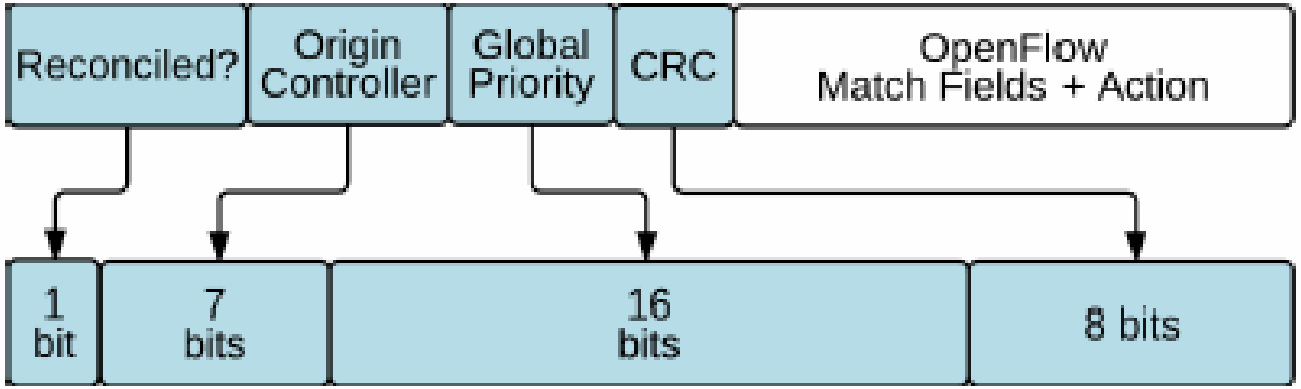
- Resolve all rules recursively to atomic actions
  - Terminal actions of Permit/Drop
- In case of action sets
  - Insert additional flow rule for each action
- Reconciliation of rules

FLOW PREPPING

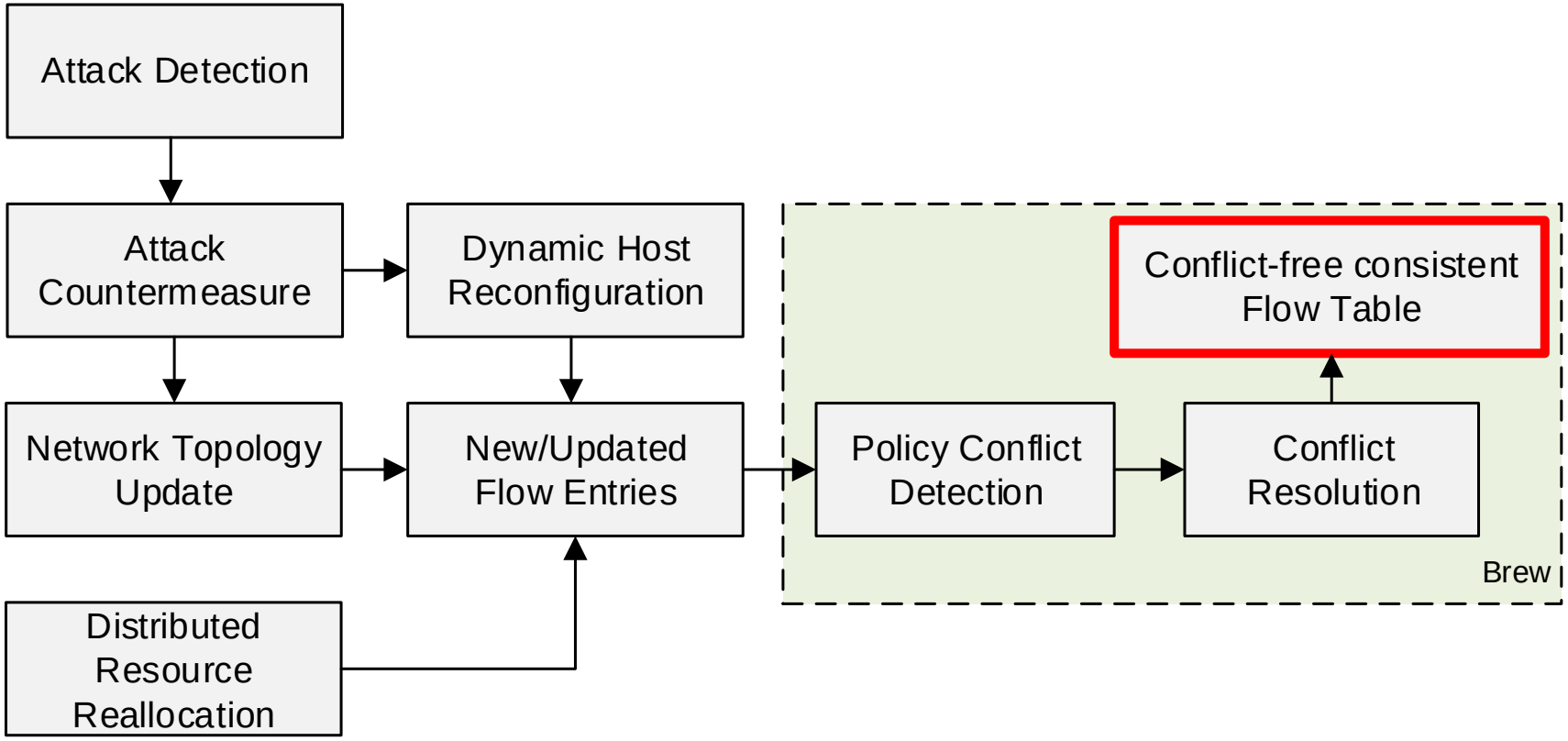
- Reconciliation of rules
  - Currently doing 1-1 mapping
  - Need to consider
    - Multiple tenant situation
    - Wildcard rules involving L2
  - Temporal mapping

SYSTEM MODULES

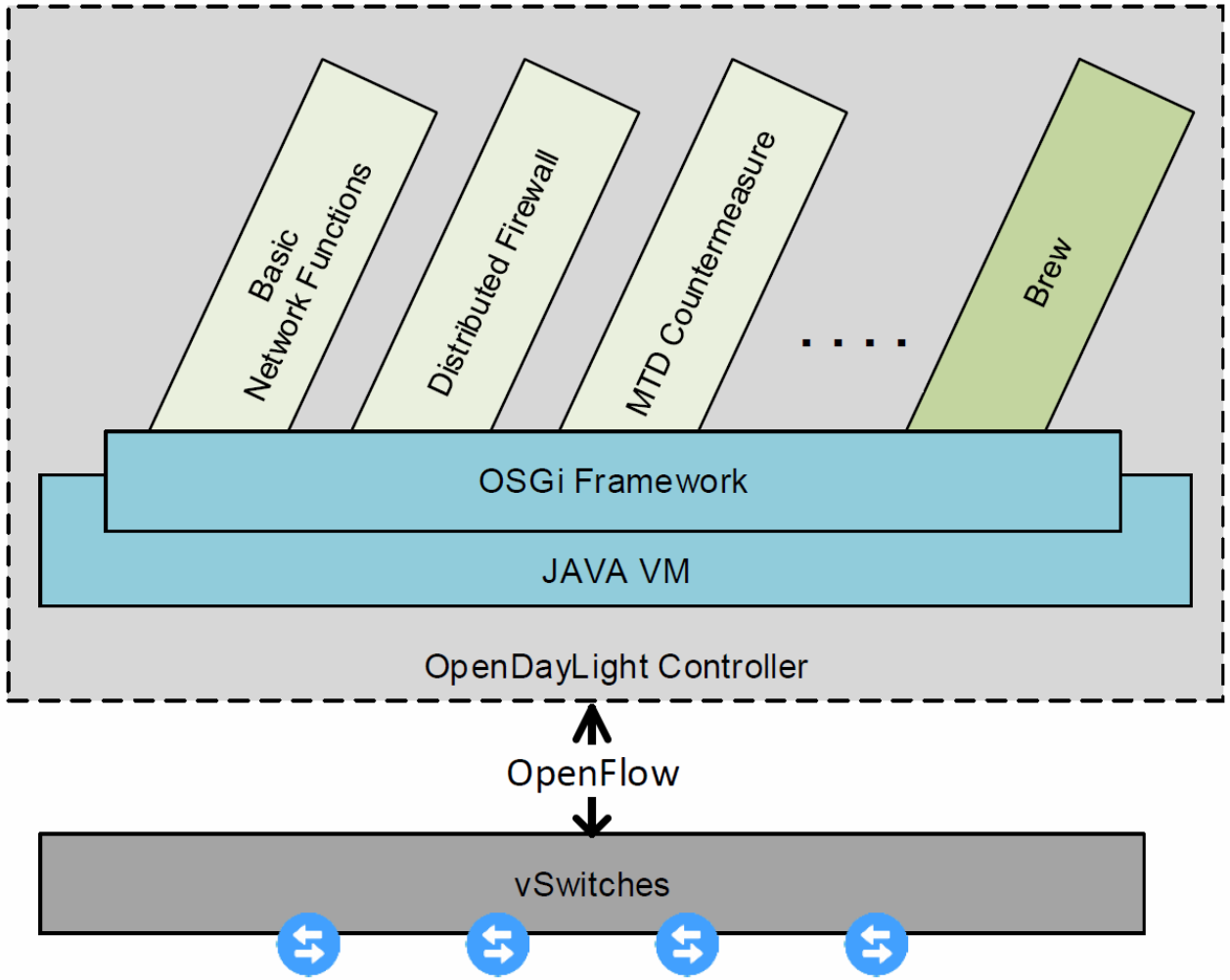
- Flow Extraction
  - Intercepts candidate flow rule that is to be added to the flow table
  - Rules from operational and configuration tree are used



BREW


Attack Detection
Attack Countermeasure
Dynamic Host Reconfiguration
Network Topology Update
New/Updated Flow Entries
Distributed Resource Reallocation
Conflict-free consistent Flow Table
Policy Conflict Detection
Conflict Resolution
Brew

BREW


Basic Network Functions
Distributed Firewall
MTD Countermeasure
. . . .
Brew
OSGi Framework
JAVA VM
OpenDayLight Controller
OpenFlow
vSwitches

GLOBAL PRIORITIES

- Host Partitioning
  - Local rules are preferred
- Hierarchical controllers
  - Local (or leaf) controllers have the lowest priority
- Application Partitioning
  - Weighted priorities depending on generating application
    - Security
    - Reliability
    - Manual ordering

SANITIZATION

- Resolve all rules recursively to atomic actions
  - Terminal actions of Permit/Drop/QoS
- In case of action sets
  - Insert additional flow rule for each action
- Reconciliation of rules
  - Currently doing 1-1 mapping

SYSTEM MODULES

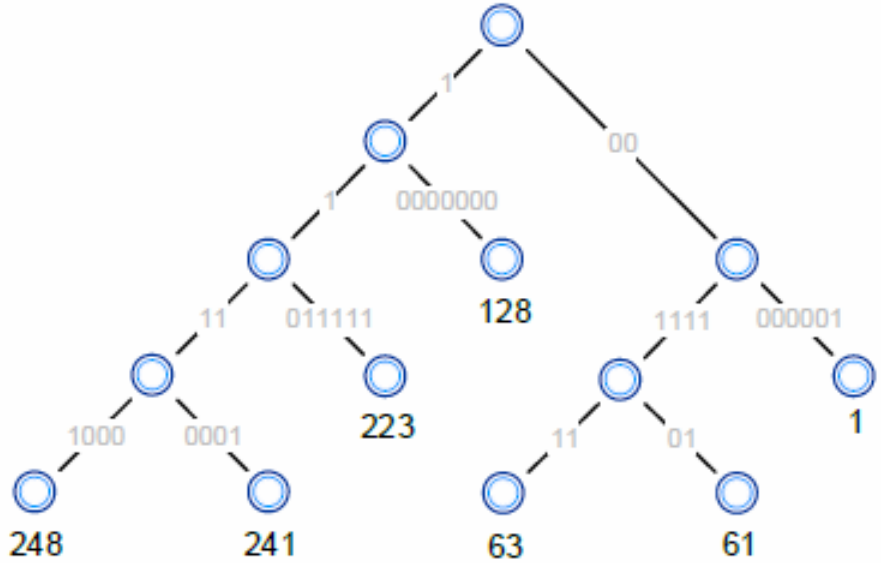
- Conflict Detection
  - Octet-wise Patricia trie lookup

Algorithm 1 Conflict Detection Module

```
1:  procedure FLOWCONFLICTS(FlowTable f)
2:  Input: Rule r, FlowTable f
3:  Output: Conflict-free FlowTable f'
4:      ATOMIZE(r)
5:      if r is layer-2 rule || layer-4 rule then
6:          RECONCILE(r)
7:      if r doesn't have Reconciled tag then
8:          γ ← SearchPatricia(L3Addr(r))
9:          if Protocol(r)! = Protocol(γ) then
10:             f' ← InsertFlow(f, r) return f'
11:         if Addr(r) ⊆ Addr(γ) then
12:             if Action(r) = Action(γ) then
13:                 ConflictResolve(r, γ, Redundancy)
14:             else
15:                 if Priority(r) == Priority(γ) then
16:                     ConflictResolve(r, γ, Correlation)
17:                 else
18:                     ConflictResolve(r, γ, Generalization)
19:         if Addr(γ) ⊆ Addr(r) then
20:             if Action(r) = Action(γ) then
21:                 ConflictResolve(r, γ, Redundancy)
22:             else
23:                 if Priority(r) == Priority(γ) then
24:                     ConflictResolve(r, γ, Correlation)
25:                 else
26:                     ConflictResolve(r, γ, Shadowing)
27:         if Addr(r) ∩ Addr(γ)! = ∅ then
28:             if Action(r) = Action(γ) then
29:                 ConflictResolve(r, γ, Overlap)
30:             else
31:                 ConflictResolve(r, γ, Correlation)
32:         Insert r to the PatriciaTrie and flow table
33:     else
34:         // Rules with Reconciled tag
35:         for Rule γ in f do
36:             if Protocol(r) == Protocol(γ) then
37:                 if Addr(r) ∩ Addr(γ)! = ∅ then
38:                     ConflictResolve(r, γ, Imbrication)
39:     Insert r to the PatriciaTrie and flow table
```

CONFLICT DETECTION

- Patricia trie data structure
  - Showing {1,61,63,128,223,241,248}



CONFLICT RESOLUTION

- Categorization
  - Tier-1 conflict (interpretative)
    - Imbrication
      - Temporal
      - Takes current system state into account
  - Tier-2 conflicts (interpretative)
    - Generalization, Correlation
      - Potential loss of information
  - Tier-3 conflicts (intelligible)

CONFLICT RESOLUTION

- Use global priorities in decision making
  - Least privilege
  - Module security precedence
    - Static policy enforcement
  - Tenant policy enforcement
  - Others
    - Software reliability
    - Environment calibrated
    - Administrator assistance

CONFLICT RESOLUTION

Algorithm 3: Conflict Resolution Engine

Input : *Rule* r, *Rule* γ, *FlowTable* f, *String* $ConflictType$
Output : Conflict-free *FlowTable* f'
Procedure `ConRes()`

```
1  if ConflictType == Shadowing || ConflictType == Redundancy then
2      return f
3  else if ConflictType == Correlation then
4      if γ.globalPriority > r.globalPriority then
5          r.addr ← r.addr − γ.addr
6          f′ ← AddFlow (f, r)
7      else
8          f′ ← RemoveFlow (f, γ)
9          γ.addr ← γ.addr − r.addr
10         f′ ← AddFlow (f, r)
11         f′ ← AddFlow (f, γ)
12 else if ConflictType == Generalization then
13     f′ ← RemoveFlow (f, γ)
14     γ.addr ← γ.addr − r.addr
15     f′ ← AddFlow (f, γ)
16     f′ ← AddFlow (f, r)
17 else if ConflictType == Overlap then
18     r.addr ← r.addr + γ.addr
19     f′ ← RemoveFlow (f, γ)
20     f′ ← AddFlow (f, r)
21 else if ConflictType == Imbrication then
22     if γ.globalPriority > r.globalPriority then
23         r.addr ← r.addr − γ.addr
24         f′ ← AddFlow (f, r)
25     else
26         f′ ← RemoveFlow (f, γ)
27         γ.addr ← γ.addr − r.addr
28         f′ ← AddFlow (f, r)
29         f′ ← AddFlow (f, γ)
30 return f′
```

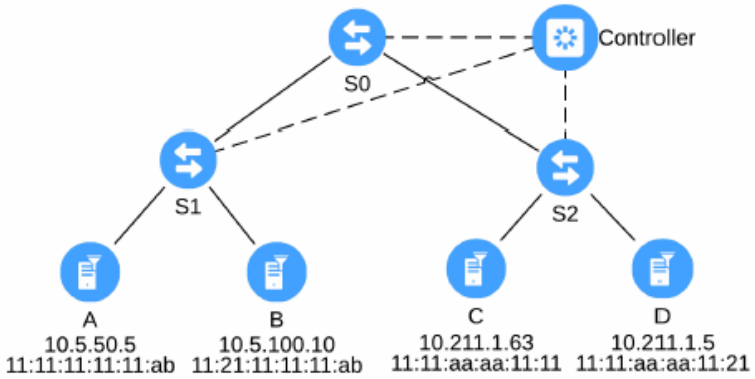
CONFLICT RESOLUTION STRATEGIES

- Intelligible conflict
  - Same action
    - Resolved without loss of information
- Interpretative conflict
  - Different actions
    - Lossy resolution

CONFLICT RESOLUTION STRATEGIES

- Assigned global priorities
  - Least privilege
  - Module security precedence
    - Static policy enforcement
  - Tenant policy enforcement
  - Others
    - Software reliability
    - Environment calibrated
    - Administrator assistance

EVALUATION



Rule #	Priority	Source MAC	Dest MAC	Source IP	Dest IP	Protocol	Source Port	Dest Port	Action
1	51	*	*	10.5.50.0/24	10.211.1.63	tcp	*	*	permit
2	50	*	*	10.5.50.5	10.211.1.63	tcp	*	80	permit
3	52	*	*	10.5.50.5	10.211.1.0/24	tcp	*	*	permit
4	53	*	*	10.5.50.0/24	10.211.1.63	tcp	*	*	deny
5	54	*	*	10.5.50.5	10.211.1.63	tcp	*	*	deny
6	51	*	*	10.5.50.0/16	10.211.1.63	tcp	*	*	deny
7	55	*	*	10.5.50.5	10.211.1.0/24	tcp	*	80-90	deny
8	57	11:11:11:11:11:ab	11:11:aa:aa:11:11	*	*	*	*	*	permit
9	58	*	*	*	*	tcp	*	80	deny

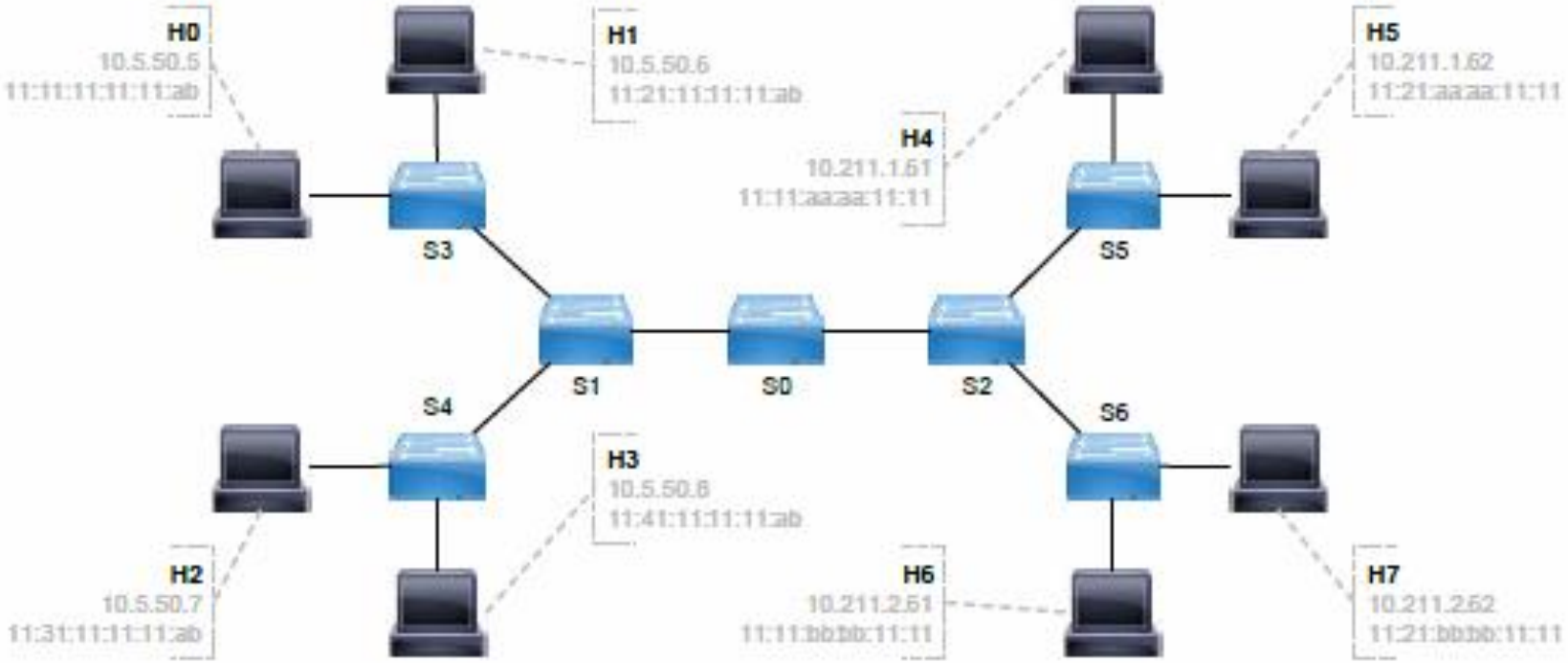
EVALUATION

- Both the conflict detection and resolution algorithms grow in a linear fashion
- Complexity of a lookup on a Patricia trie
  - a total runtime of O(n)
- Verified on an input file containing about 10,000 atomic flow rules
  - processing time was about 6:45 ms.

EVALUATION

- Both the conflict detection and resolution algorithms grow in a linear fashion
- Complexity of a lookup on a Patricia trie
  - a total runtime of O(n)
- Verified on an input file containing about 100,000 atomic flow rules
  - processing time was about 56 ms

CORRECTNESS

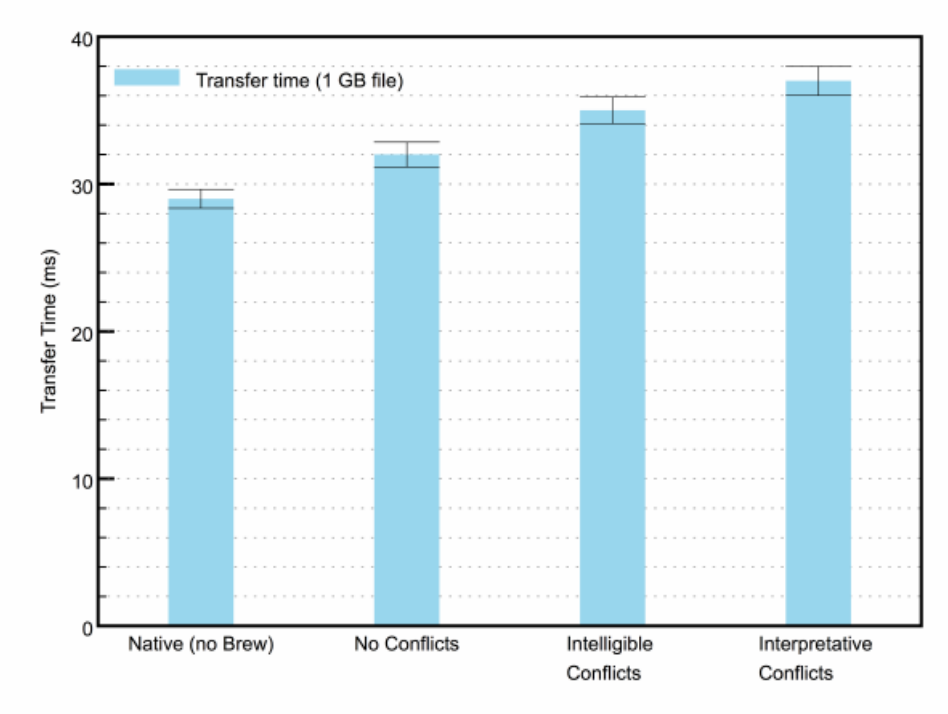


CORRECTNESS

Rule #	Priority	Source MAC	Dest MAC	Source IP	Dest IP	Protocol	Source Port	Dest Port	Action
1	51	*	*	10.5.50.0/24	10.211.1.63	tcp	*	*	forward
2	50	*	*	10.5.50.5	10.211.1.63	tcp	*	80	forward
3	52	*	*	10.5.50.5	10.211.1.0/24	tcp	*	*	forward
4	53	*	*	10.5.50.0/24	10.211.1.63	tcp	*	*	drop
5	54	*	*	10.5.50.5	10.211.1.63	tcp	*	*	drop
6	51	*	*	10.5.50.0/16	10.211.1.63	tcp	*	*	drop
7	55	*	*	10.5.50.5	10.211.1.0/24	tcp	*	1000-1007	drop
8	57	11:11:11:11:11:ab	11:11:aa:aa:11:21	*	*	*	*	*	forward
9	58	*	*	*	*	tcp	*	80	drop

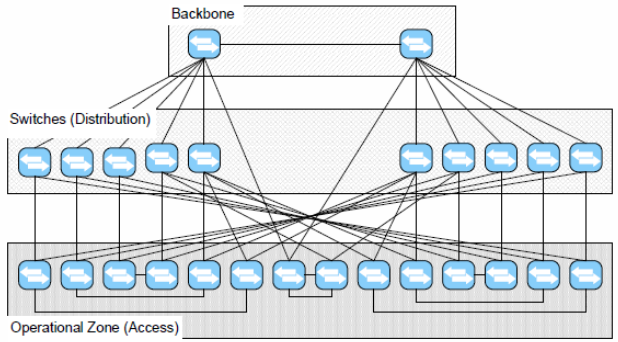
OVERHEAD

- File transfer between 2 nodes, with and without Brew

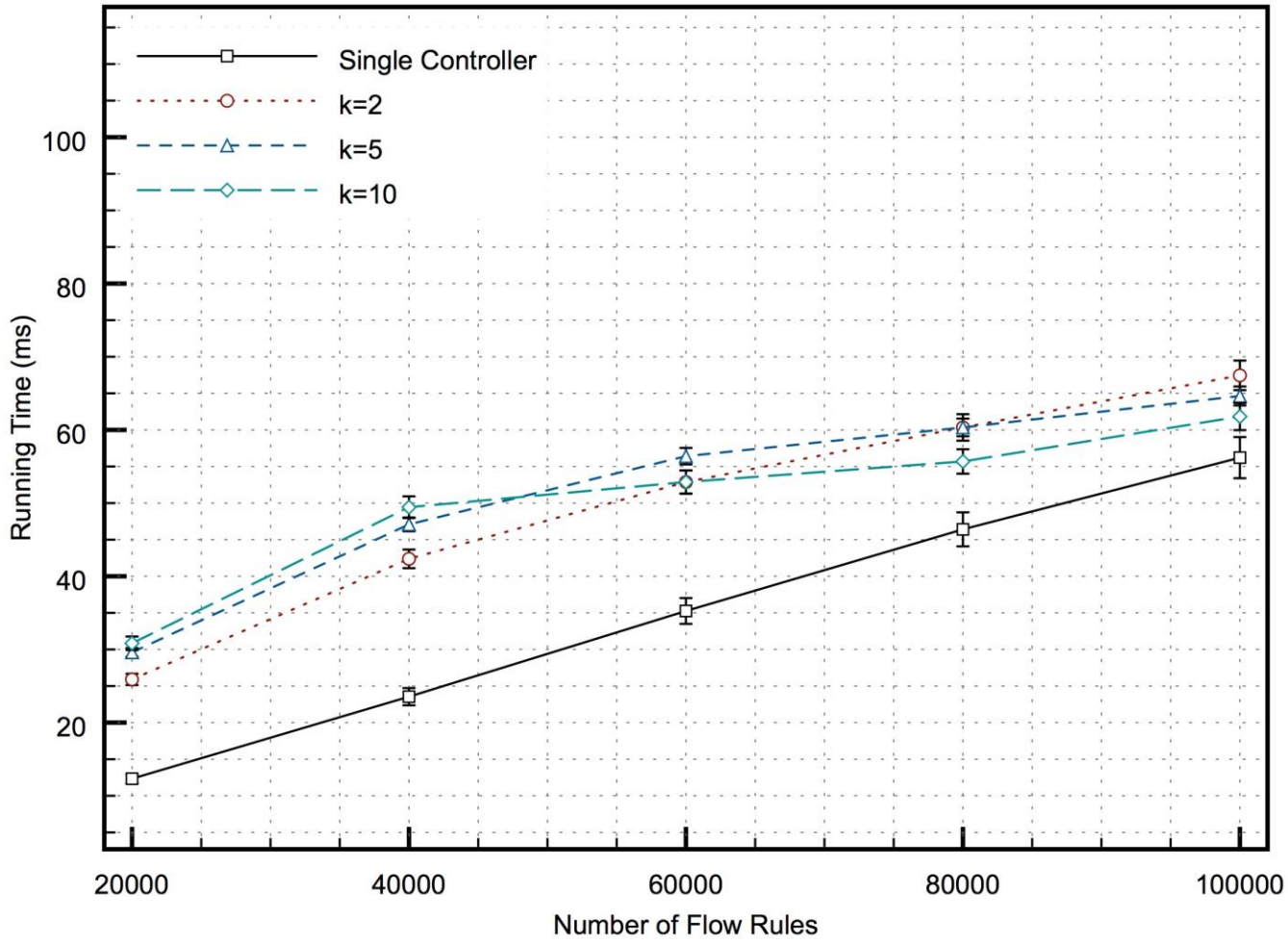


SCALABILITY

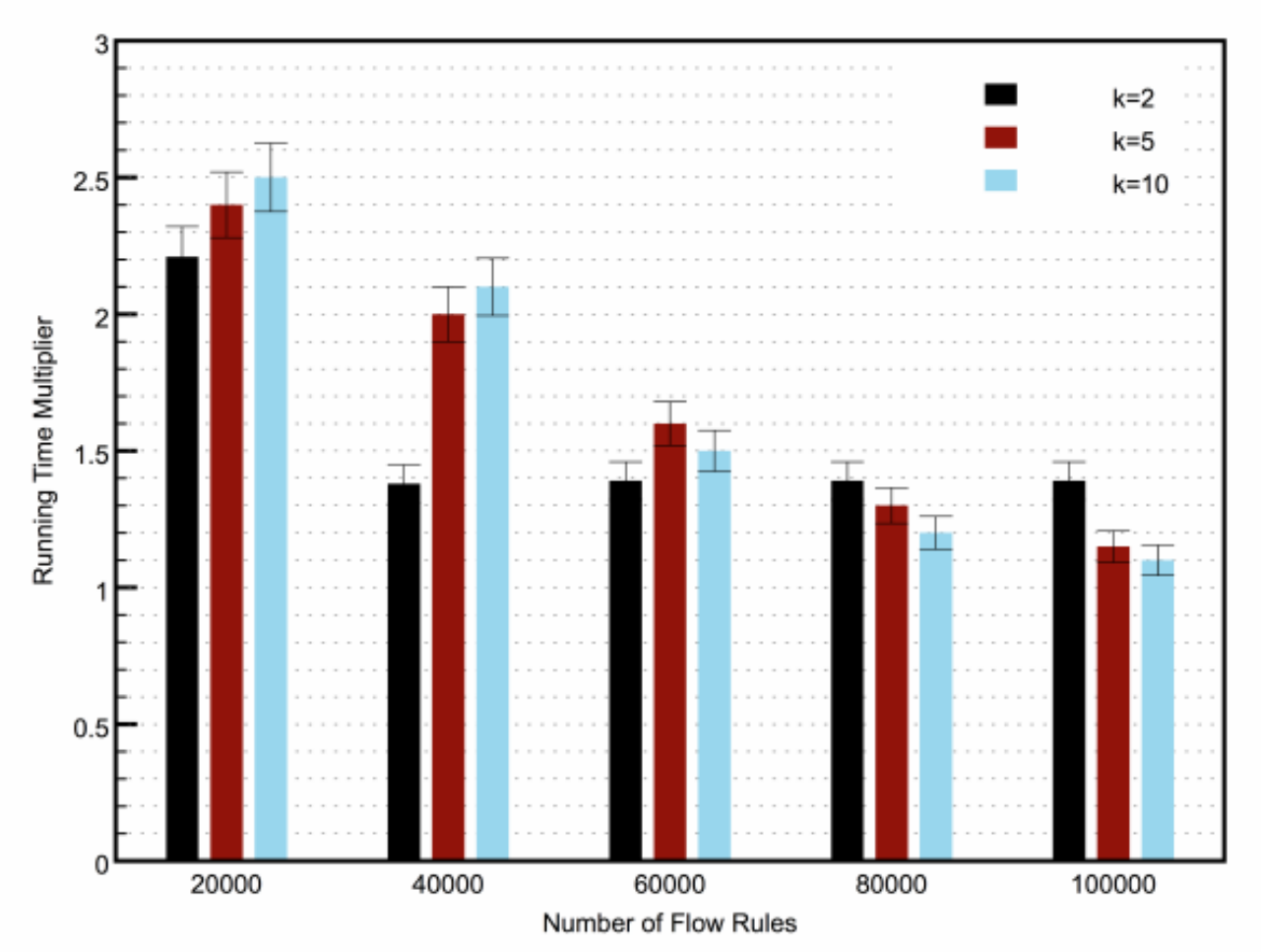
- Stanford topology
  - Snapshot of the backbone configurations
    - 12,900 routes, 757,000 forwarding entries, 100 VLANs and 900 access-list rules
    - Replicated in Mininet
      - Approximately 8,900 atomic flow rules
        - 578 conflicts (~6.5%)
          - 431 overlaps
        - No imbrication conflicts
      - Extrapolated to 100K



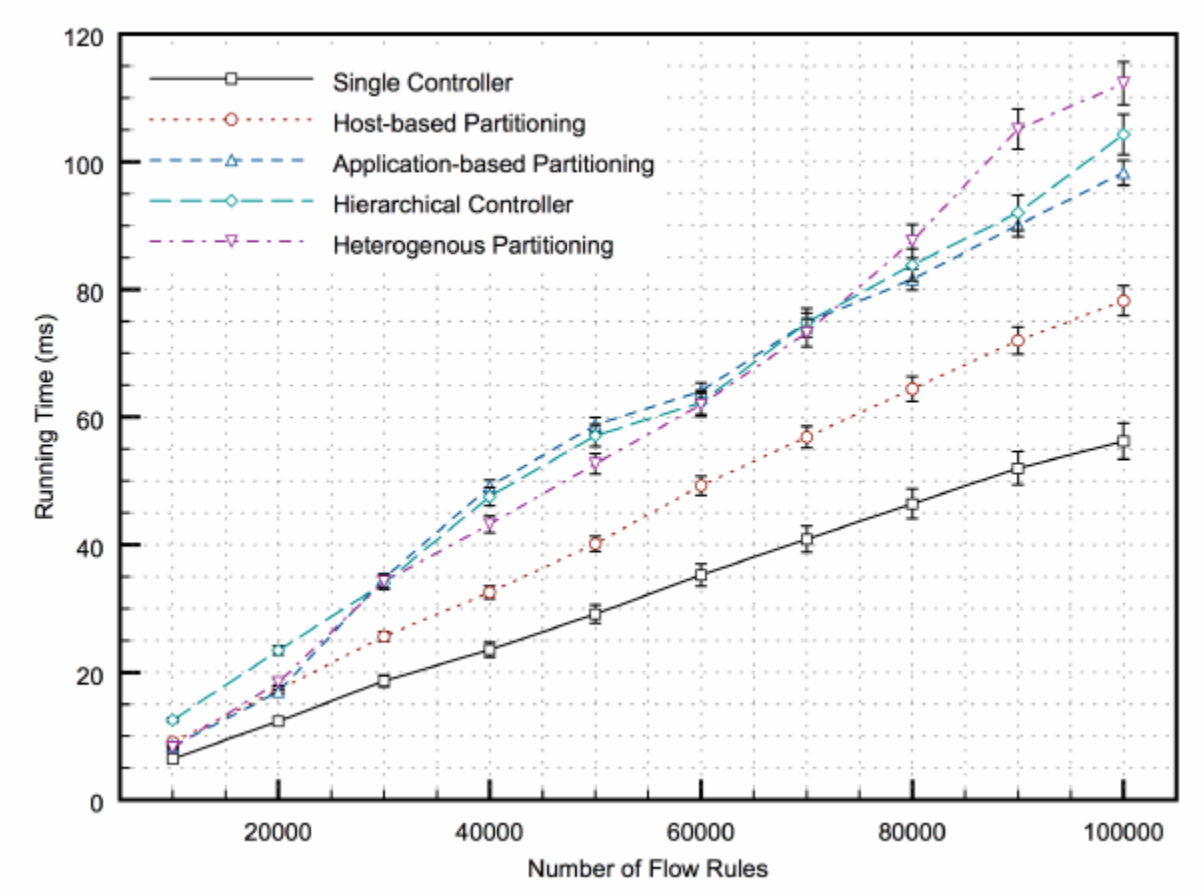
MULTIPLE CONTROLLERS



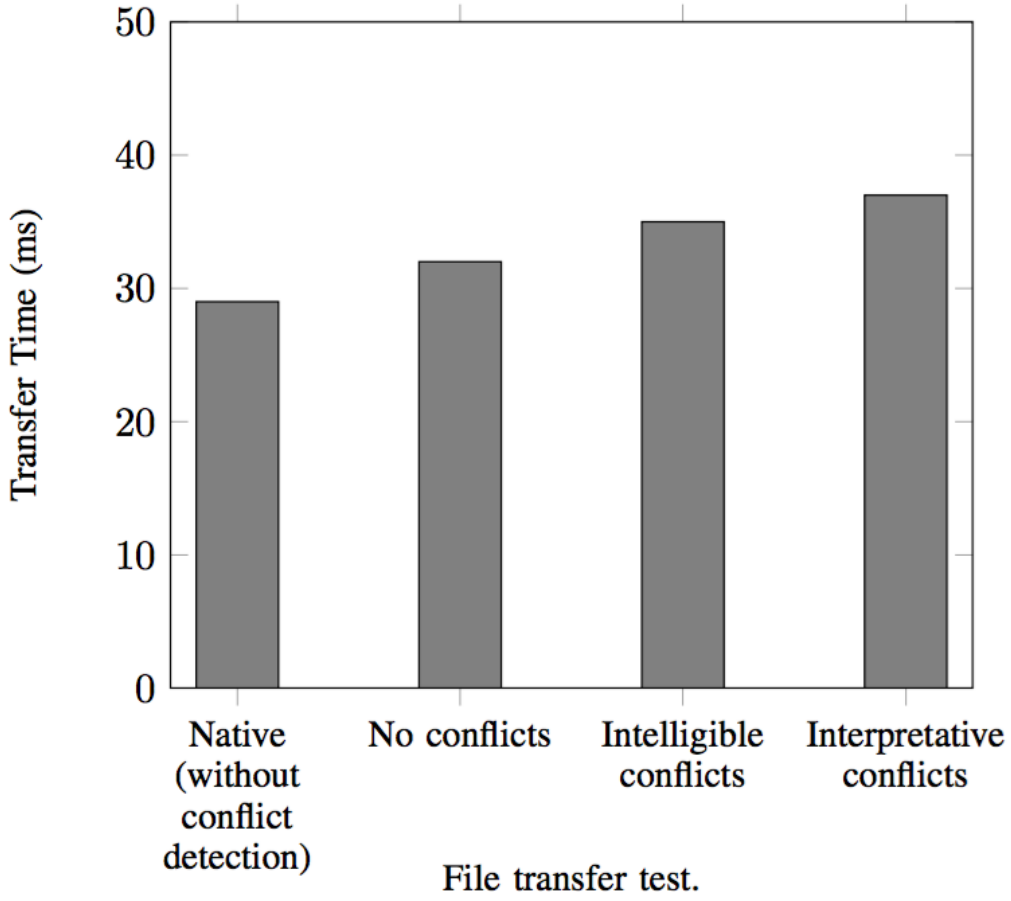
MULTIPLE CONTROLLERS



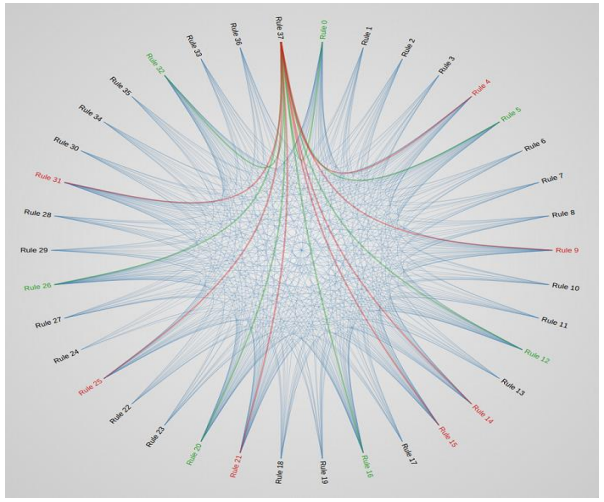
PARTITIONING STRATEGY

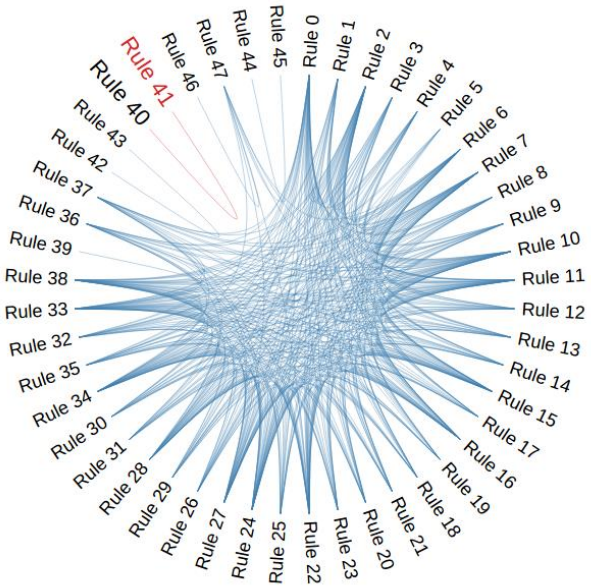


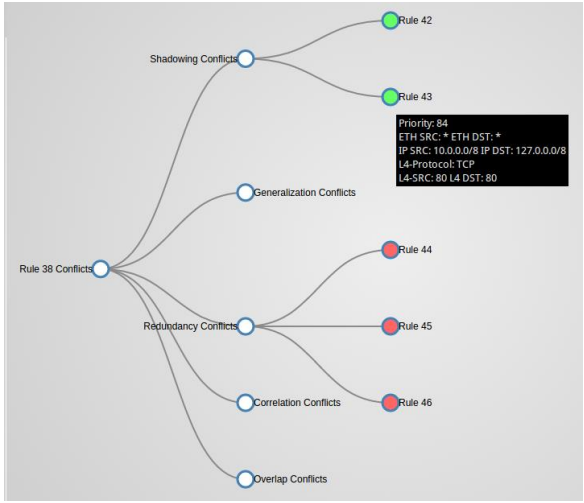
OVERHEAD


50
40
30
20
10
0
Transfer Time (ms)
Native (without conflict detection)
No conflicts
Intelligible conflicts
Interpretative conflicts
File transfer test.

VISUALIZATION







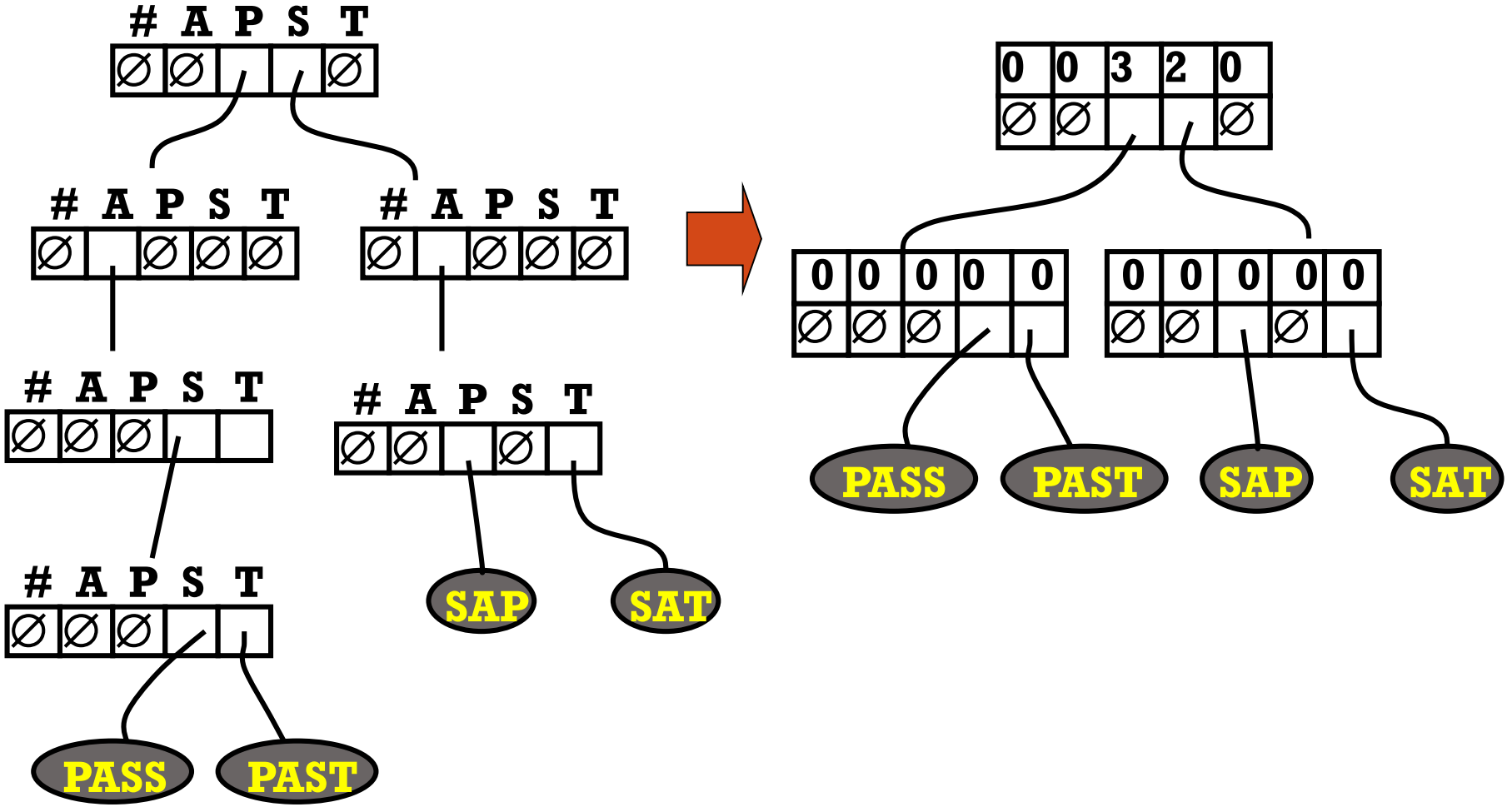
PATRICIA TRIE

- Data structure for storing a set of strings.
- Each edge of the tree is labeled with a character.
- Each leaf node corresponds to the stored string, which is a concatenation of characters on a path from the root to this node.

PATRICIA TRIE

- Collapse chains of nodes that have only one child
- For each branch indicate how many characters should be skipped (i.e. what the length of the collapsed chain is)

PATRICIA TRIE

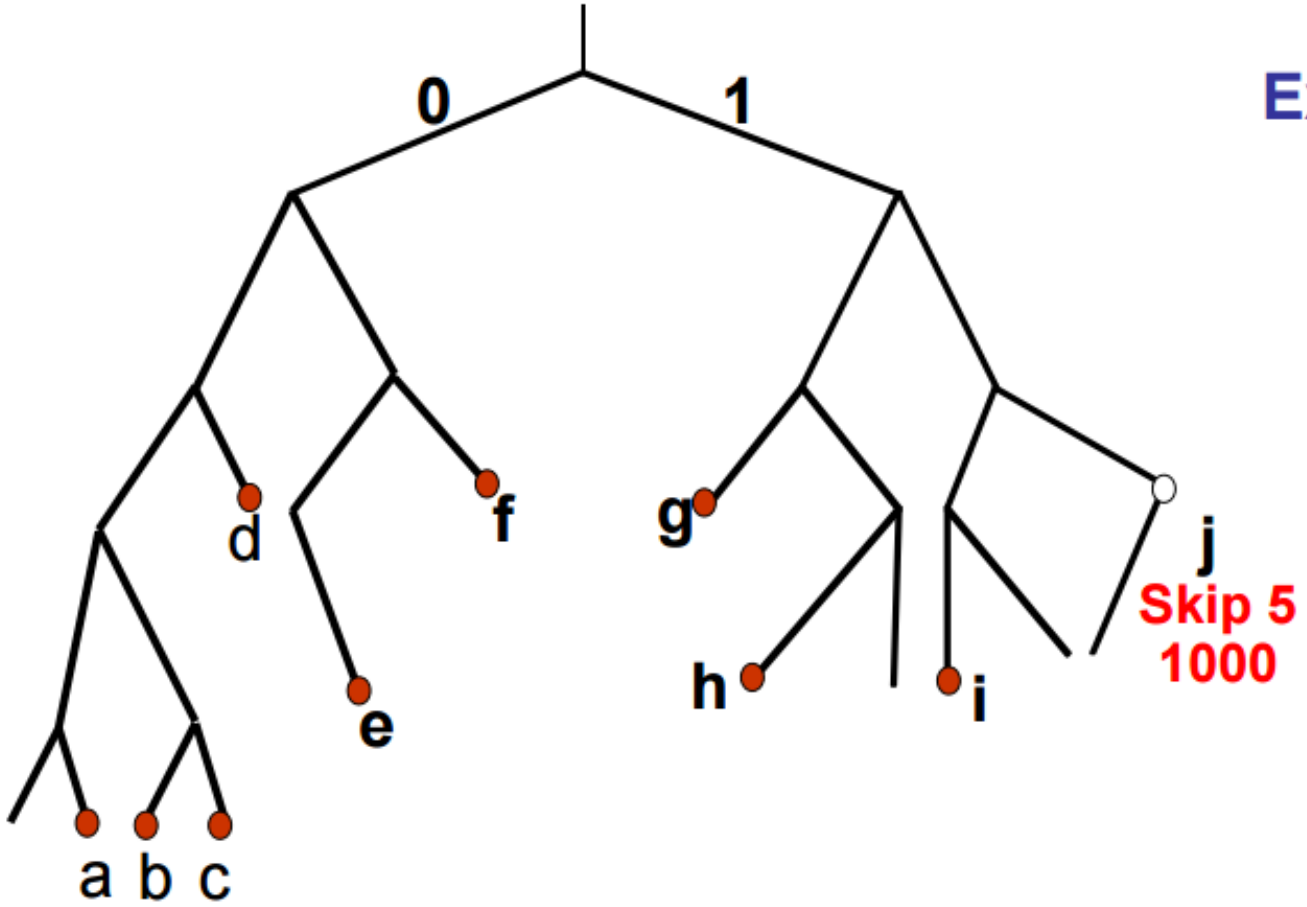


PATRICIA TRIE

- Why Patricia Trie?
  - Faster than linear scan
    - Proportional to number of bits in the address
  - Trie is not high
  - Space overhead problem is not valid when we know the maximum storage required

Left-ptr Bit # Right-ptr

PATRICIA TRIE



Example Prefixes

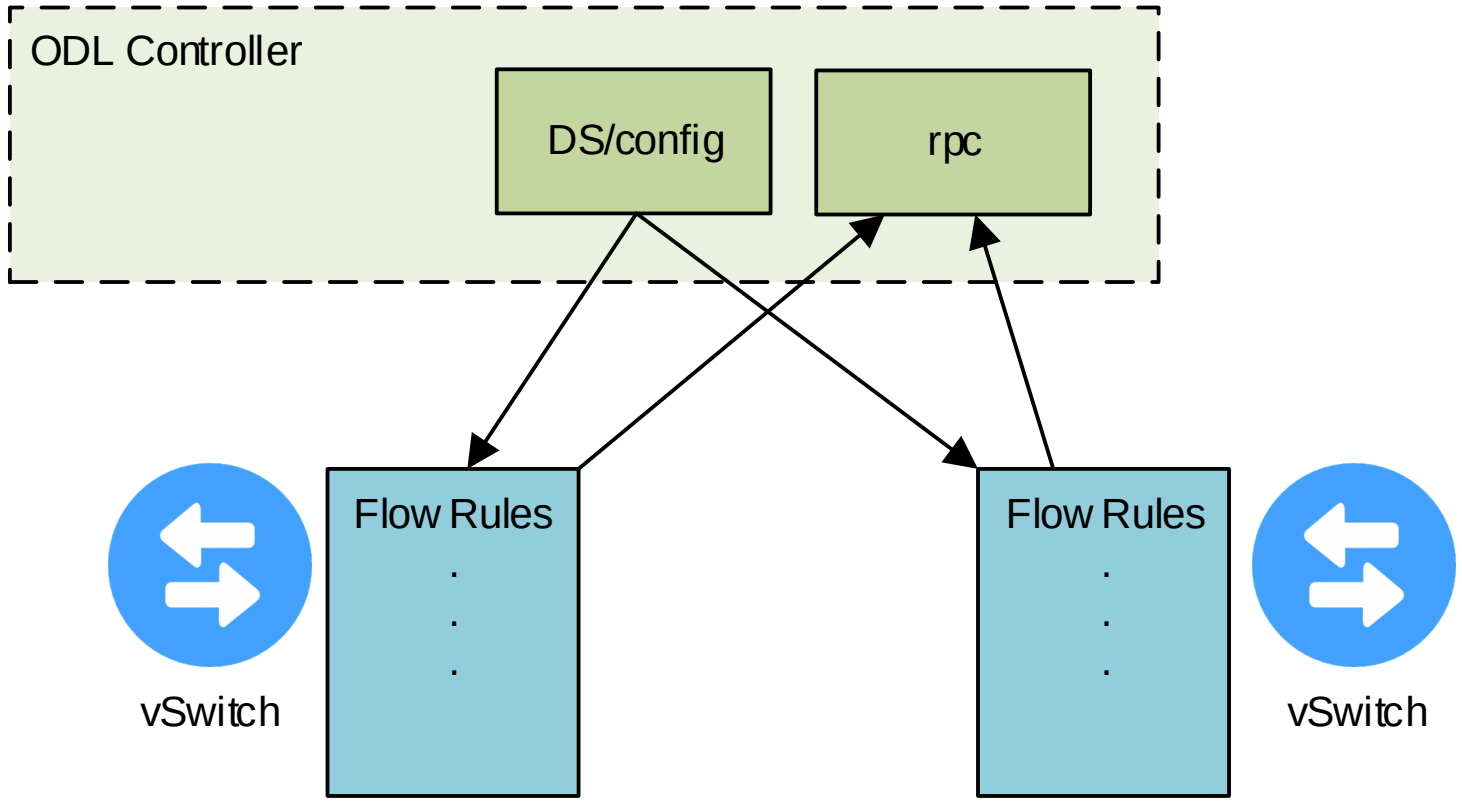
a) 00001
b) 00010
c) 00011
d) 001
e) 0101
f) 011
g) 100
h) 1010
i) 1100
j) 11110000

COMPLEXITY

- D-bit prefixes: O(D) lookup, O(ND) storage and O(D) update complexity
  - Since tree is binary, average tree height for N keys is O (log D)

- Worst Case:
  - O (ND), where N is the number of flow entries and D is the number of prefix bits

OFANALYZER


ODL Controller
DS/config
rpc
Flow Rules
Flow Rules
vSwitch
vSwitch

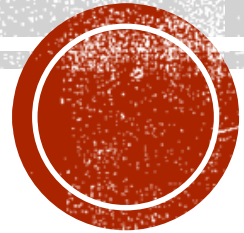
FLOW RULE MONITOR

- DS/config
  - flows can be preconfigured
  - can be stored (thus survive restart)
  - no support for bulk operations based on masks
  - unable to accurately determine if push succeeded

FLOW RULE MONITOR

- rpc
  - can make use of bulk operations based on mask
  - can *eventually* read flow on device which comes from DS/config
  - returns response
  - ability to delete whole table in one step!

RELATED WORK



RELATED WORKS

- Traditional environments
  - Firewall Policy Advisor (FPA)
    - Single rooted policy tree
  - Fireman
    - Detects inter- and intra-firewall inconsistencies
  - Language-based
    - REI, Ponder, Firmato

RELATED WORKS

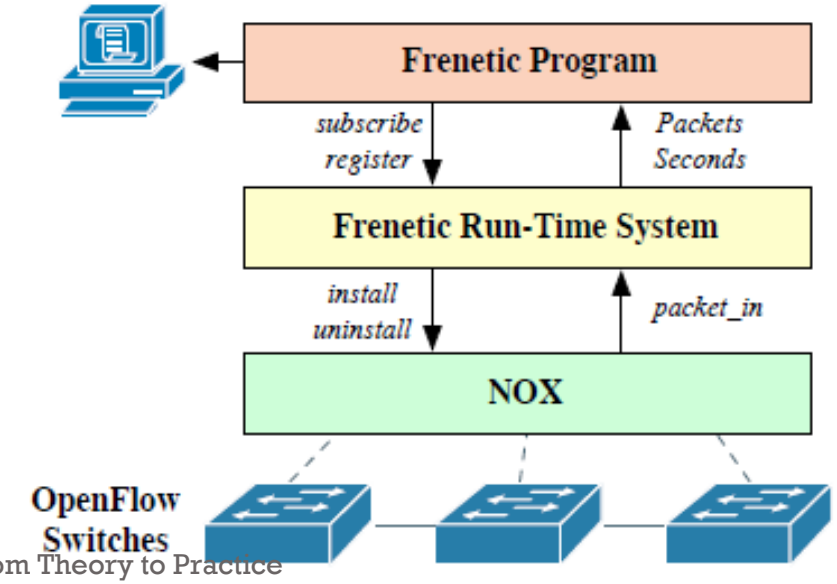
- FRESCO/FortNOX
  - Rule source identification
  - Applies a "lock" on rules placed by security applications
- VeriFlow
  - Layer between the controller and switches
    - Conducts real time verification of rules
    - Ensure no hardware failures, reachability issues

RELATED WORKS

- Frenetic
  - Network programming language for defining high level policies
- Procera
  - Formal language based
- FlowGuard
  - Examines incoming policy updates and determines violations in addition to performing stateful monitoring

FRENETIC

- Network programming language for defining high level policies
  - Provides a high-level abstraction of network functions



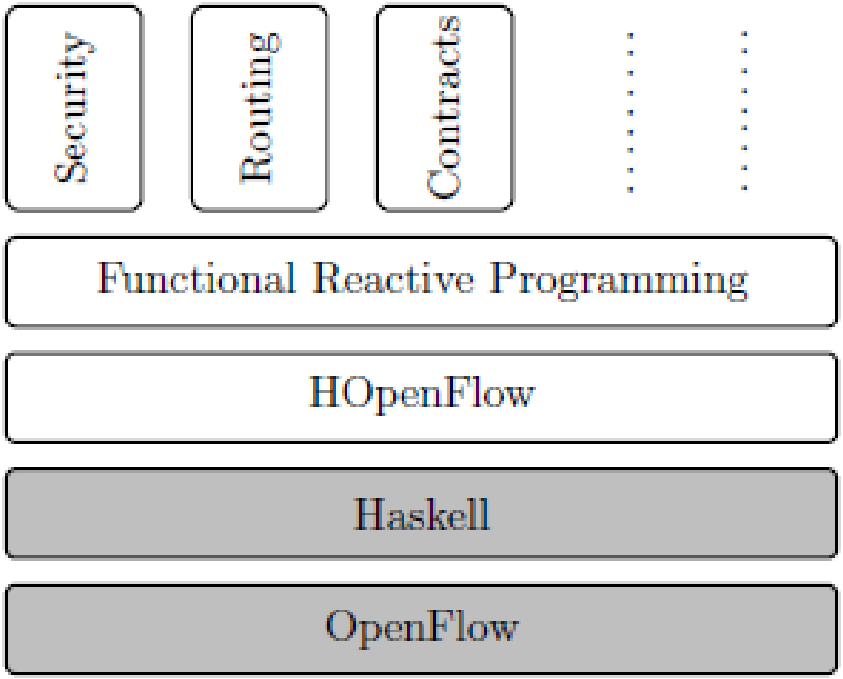
FRENETIC

- Uses Python library
- Based on Functional Reactive Programming (FRP), so ECA
- Program parses every packet
  - Will **NOT** scale to networks of realistic size

NETTLE

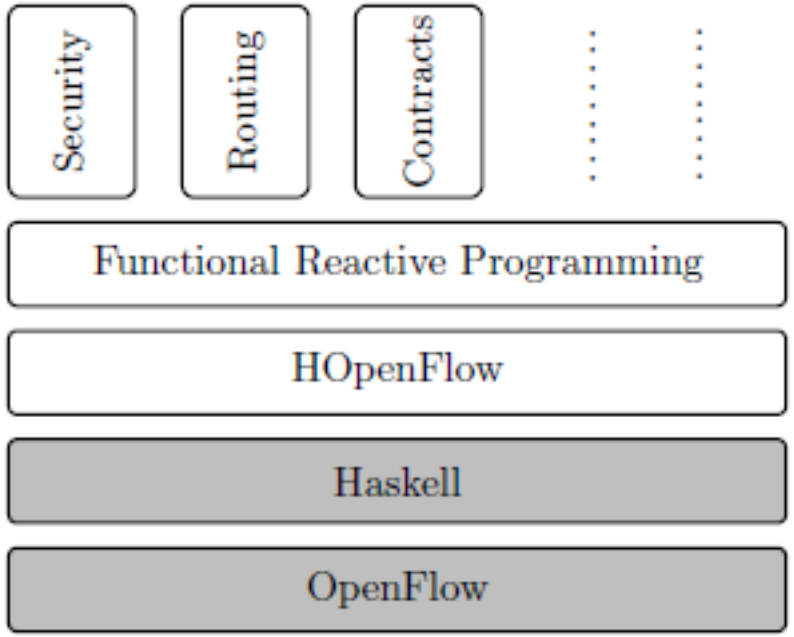
- Like Frenetic, Nettle is based on FRP
- Supports network-wide control and domain-specific languages for different tasks
- Lacks Frenetic's support for overlapping module actions
  - Nettle appears to be more of a replacement for NOX

NETTLE




```
sf2 :: (HasSwitchEvents i, HasConsoleOutput o, HasSwitchCommands o) ⇒ SF i (Event o)
sf2 = proc i → do
  returnA ⤙ packetInE i ⋙ λe → consoleOut (show e) ⊕ sendReceivedPacket e [flood]
```

NETTLE

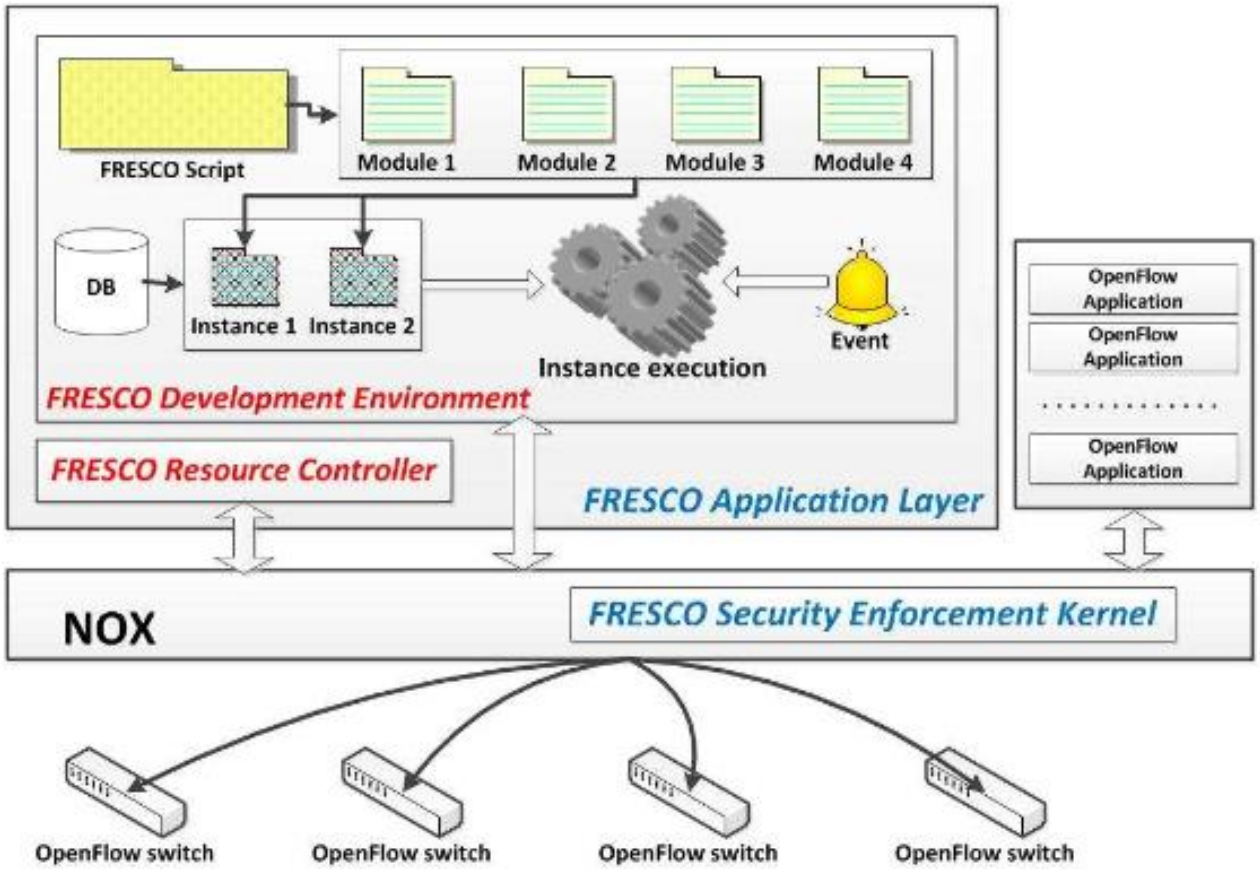


$$sf2 :: (HasSwitchEvents\ i, HasConsoleOutput\ o, HasSwitchCommands\ o) \Rightarrow SF\ i\ (Event\ o)$$
$$sf2 = \mathbf{proc}\ i \rightarrow \mathbf{do}$$
$$\quad returnA \prec packetInE\ i \ggg \lambda e \rightarrow consoleOut\ (show\ e) \oplus sendReceivedPacket\ e\ [flood]$$

FRESCO

- Allows the deployment of security services for OpenFlow
- Implemented as an application built on the NOX controller
- Comprises of Python scripts and API that allows the development of security services

FRESCO


FRESCO Script
Module 1
Module 2
Module 3
Module 4
DB
Instance 1
Instance 2
Instance execution
Event
FRESCO Development Environment
FRESCO Resource Controller
FRESCO Application Layer
OpenFlow Application
OpenFlow Application
OpenFlow Application
NOX
FRESCO Security Enforcement Kernel
OpenFlow switch
OpenFlow switch
OpenFlow switch
OpenFlow switch

FRESCO

- Modules
  - Python objects that have input, output, action, event and parameters
- Development environment (DE)
  - Converts scripts into modules
  - Database management
    - Shares info across modules
  - Event management
  - Instance execution

FRESCO

- In addition to permit, drop and modify; set action enables rewriting of packet header fields.
  - Helps implement redirect, mirror and quarantine
  - Enables to isolate suspected malicious hosts / traffic

- Rule source identification
  - Allows applications to digitally sign flow rules enabling SEK to know if the origin of each flow

FRESCO

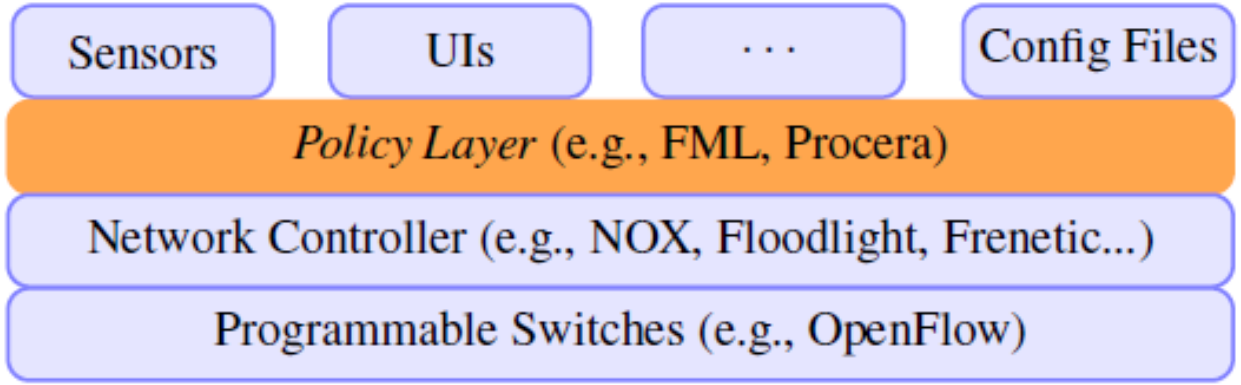
- Rule conflict detection
  - Inline rule conflict analysis algorithm (details needed)
  - Conflicts are resolved using a hierarchical authority model (based on origin as determined by digital signatures)

FRESCO

- Security Enforcement Kernel
  - Also the FortNOX project

- Applies a "lock" on rules placed by security applications
  - No rules which conflict can be inserted

PROCERA




```
proc world → do
  returnA ⊣≺
    λreq → if destIP req `inSubnet` ipAddr 128 36 5 0 // 24
           then allow else deny
```

PROCERA

- Based on FRP, like Nettle and Frenetic
  - Based on Haskell like Nettle

- Incorporates events that originate from sources other than ovSwitch
  - user authentications
  - time of day
  - bandwidth use
  - server load

PROCERA

- Very similar to Frenetic
  - Can Frenetic take non-packet inputs?

VERIFLOW

- Layer between the controller and switches
  - Conducts real time verification of rules
  - Ensure no hardware failures, reachability issues

SUMMARY

- Implementing consistent and conflict-free security policies in SDN environment is progressively challenging.
- This work formalizes, detects and resolves conflicts in a multiple controller environment.

CITE THIS WORK

```
@book{huang2018software,
title={Software-Defined Networking and Security: From Theory to Practice},
author={Huang, Dijiang and Chowdhary, Ankur and Pisharody, Sandeep},
year={2018},
publisher={CRC Press}}
```


REFERENCES

- E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 10, pp. 2069–2084, 2005.
- H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "FLOWGUARD: Building robust firewalls for software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 97–102.
- C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker *et al.*, "Composing software defined networks." in *NSDI*, 2013, pp. 1–13.
- D. R. Morrison, "Patricia - Practical algorithm to retrieve information coded in alphanumeric," *Journal of the ACM (JACM)*, vol. 15, no. 4, pp. 514–534, 1968.
- K. Poornaselvan, S. Suresh, C. D. Preya, and C. Gayathri, "Efficient IP lookup algorithm," *Strengthening the Role of ICT in Development*, p. 111, 2007.
- P. Gupta and N. McKeown, "Algorithms for packet classification," *Network, iEEE*, vol. 15, no. 2, pp. 24–32, 2001.
- F. B. Schneider, "Least privilege and more," in *Computer Systems*. Springer, 2004, pp. 253–258.