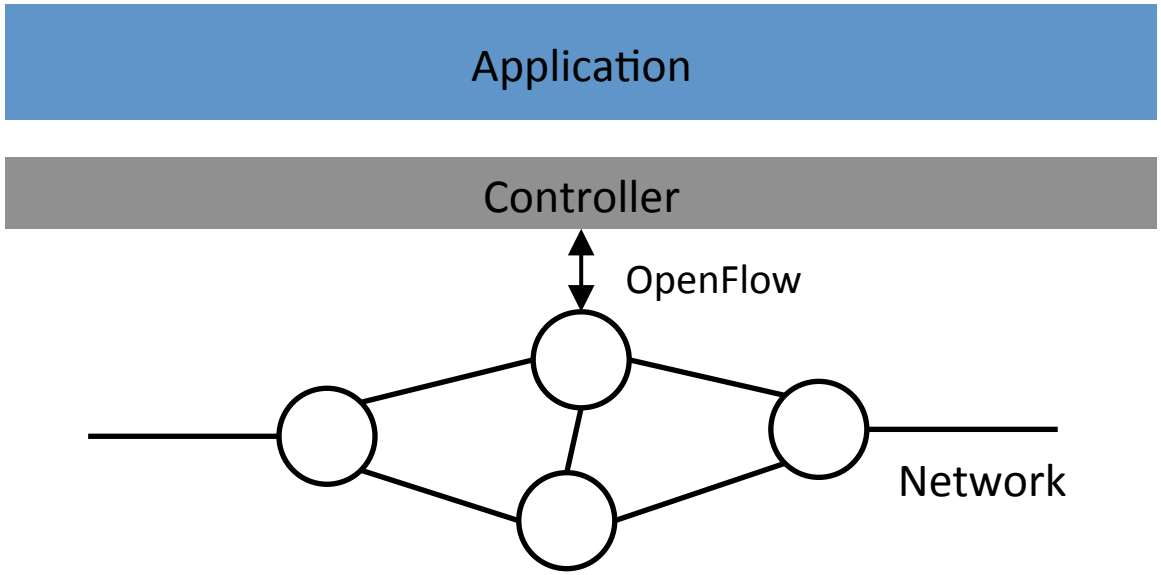
Incremental Update for a Compositional SDN Hypervisor

Xin Jin

Jennifer Rexford, David Walker

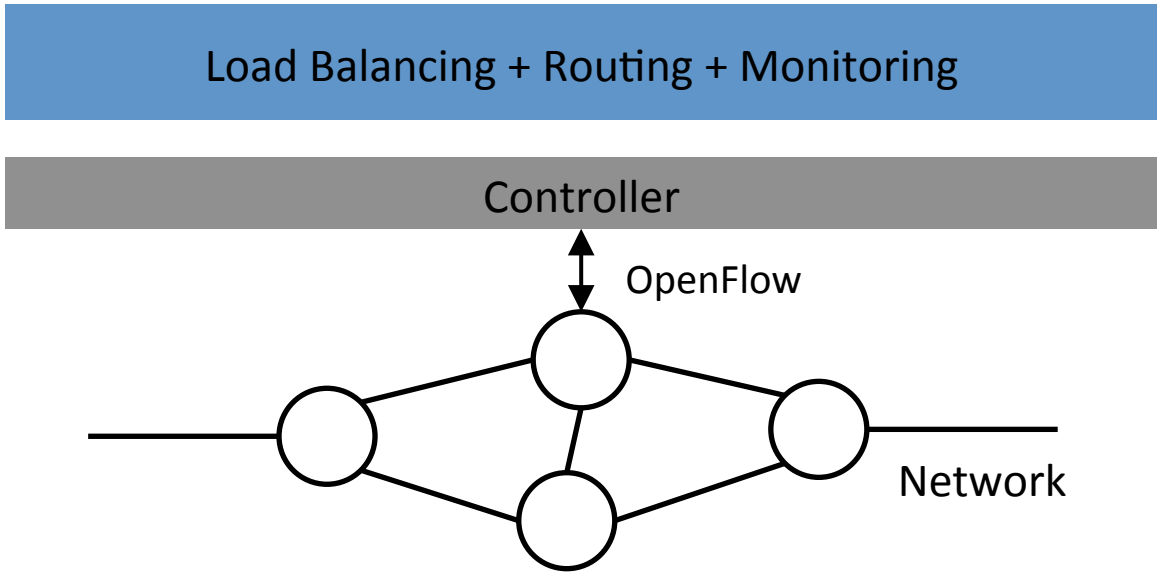
Software-Defined Networking

- Centralized control with open APIs



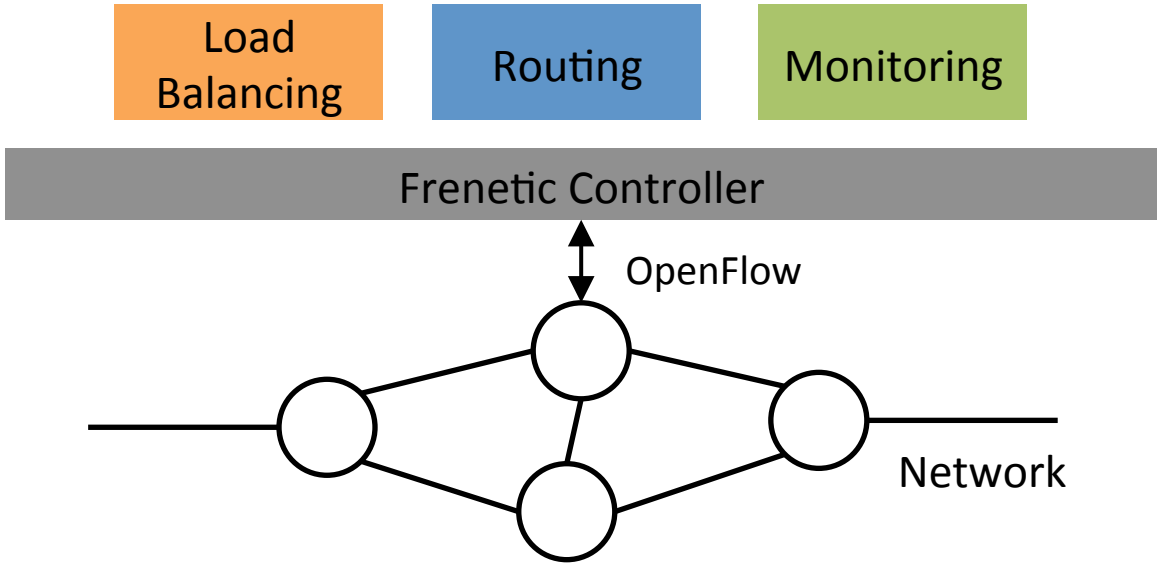
Multiple Management Tasks

- Hard to develop and maintain a monolithic application



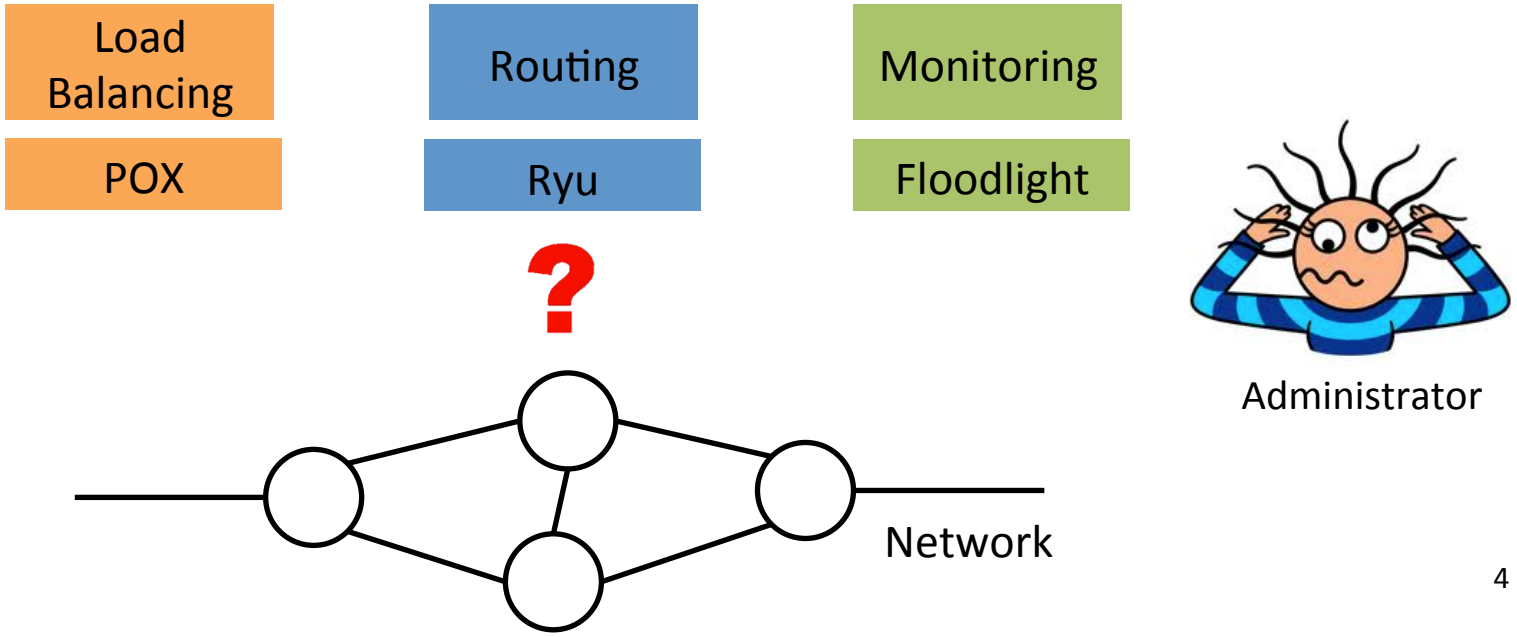
Modular SDN Applications

- Frenetic: composition operators to combine multiple applications
- Limitation: need to adopt Frenetic language and runtime system



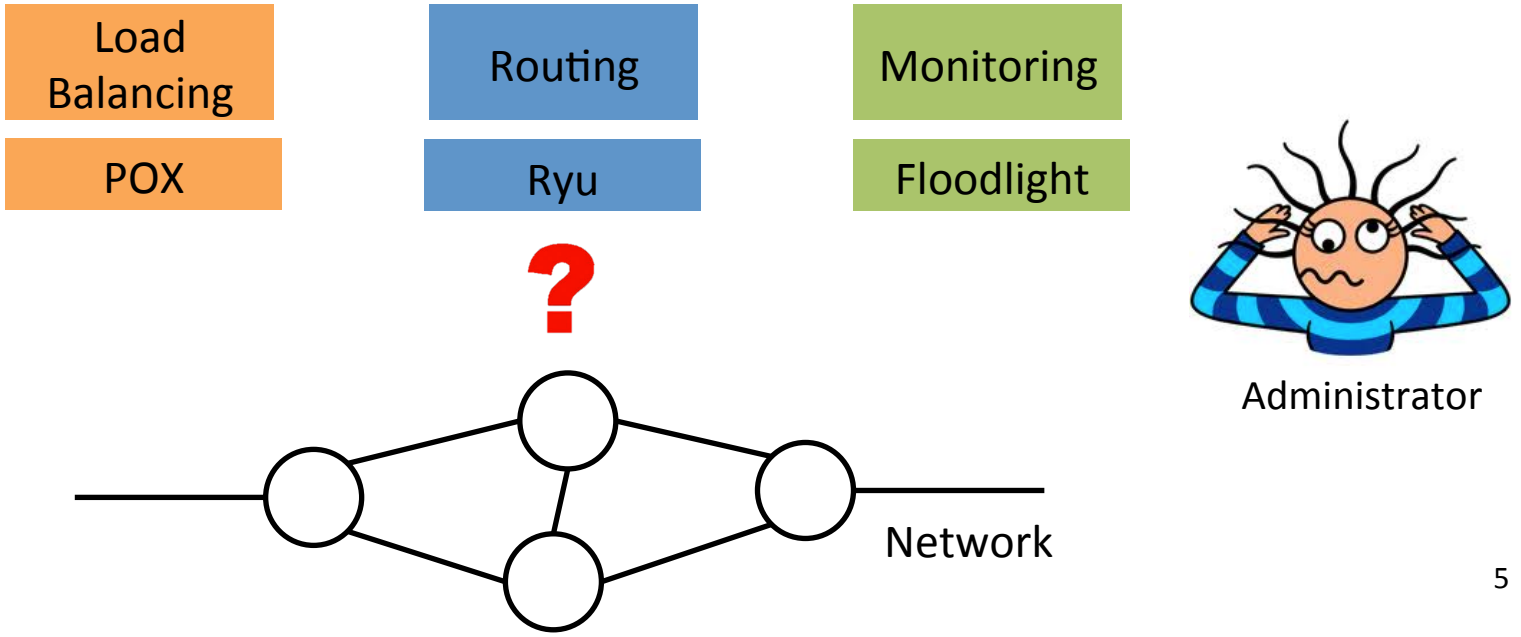
Frenetic is Not Enough

- "Best of breed" applications are developed by different parties
  - Use different programming languages
  - Run on different controllers



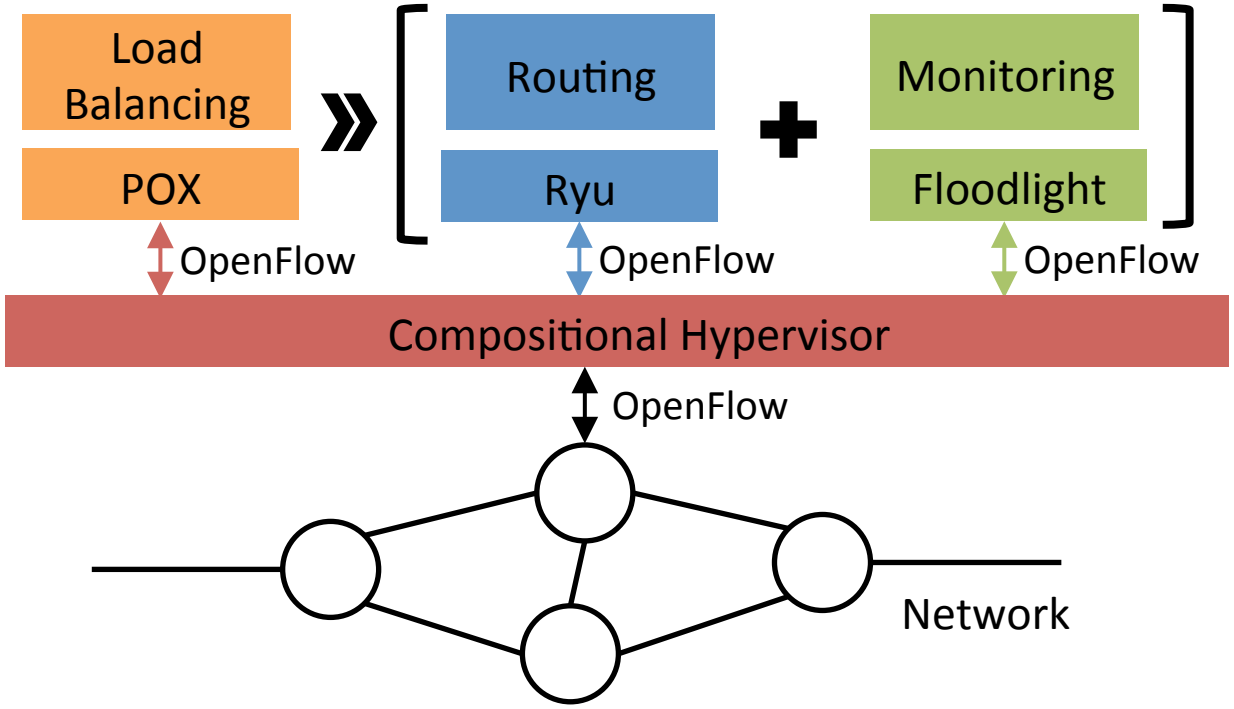
Slicing is Not Enough

- Controllers work on disjoint parts of traffic
  - Useful for multi-tenancy
- But we want them to collaboratively work on the same traffic



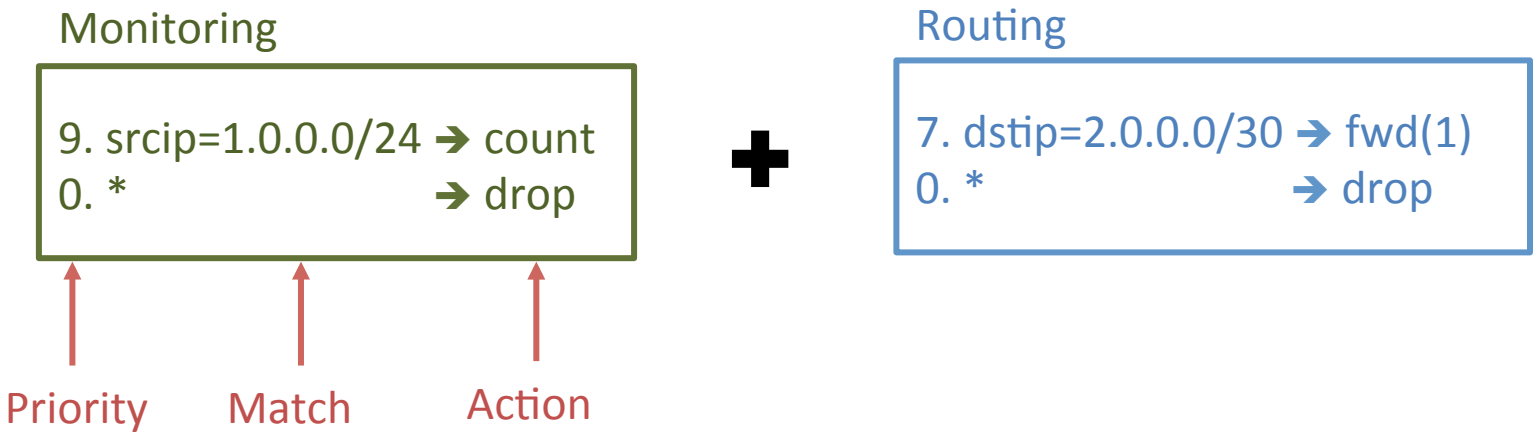
Our Solution: Compositional Hypervisor

- A transparent layer between switches and controllers
- Combine controllers with Frenetic-like composition operators
  - Parallel operator (+)
  - Sequential operator (>>)



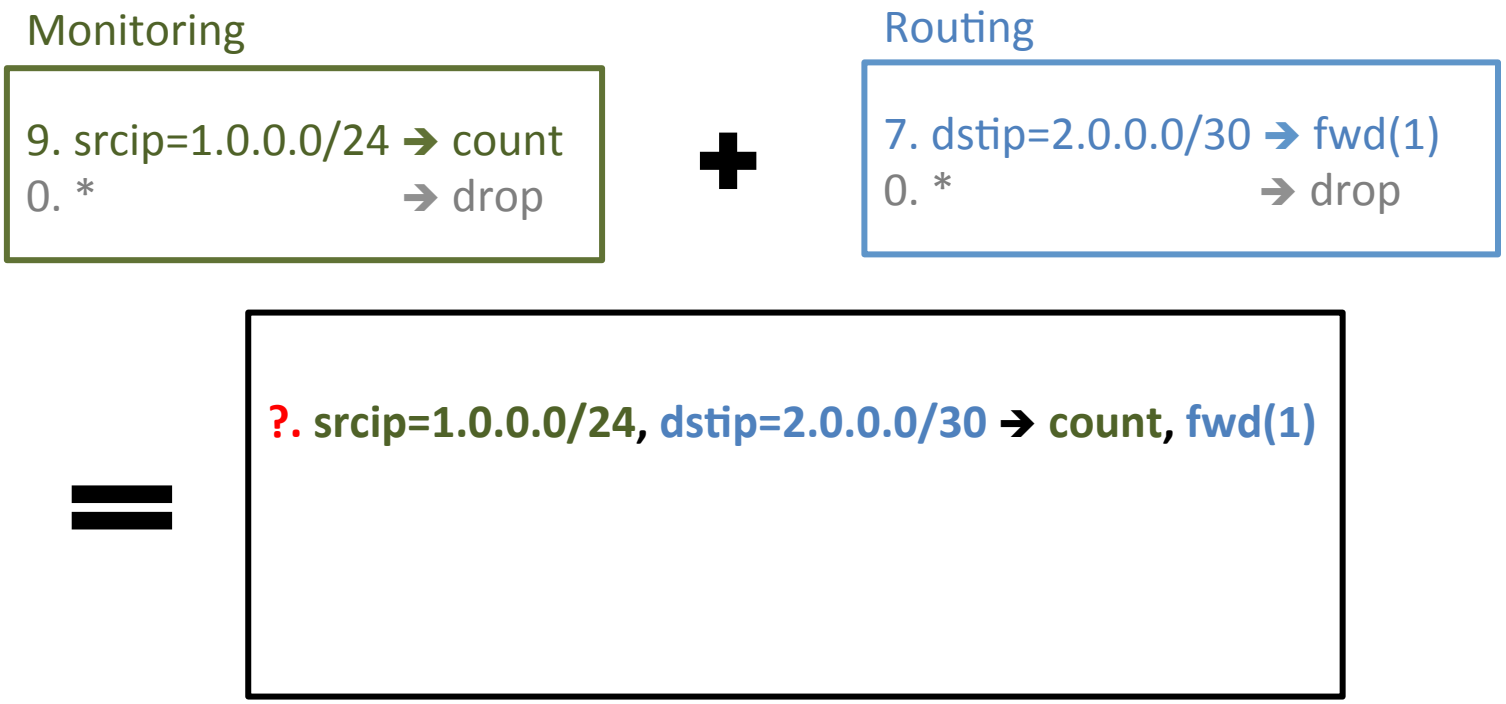
Policy Compilation

- Policy: a list of rules
- Each controller outputs a policy
- Hypervisor compiles them to a single policy



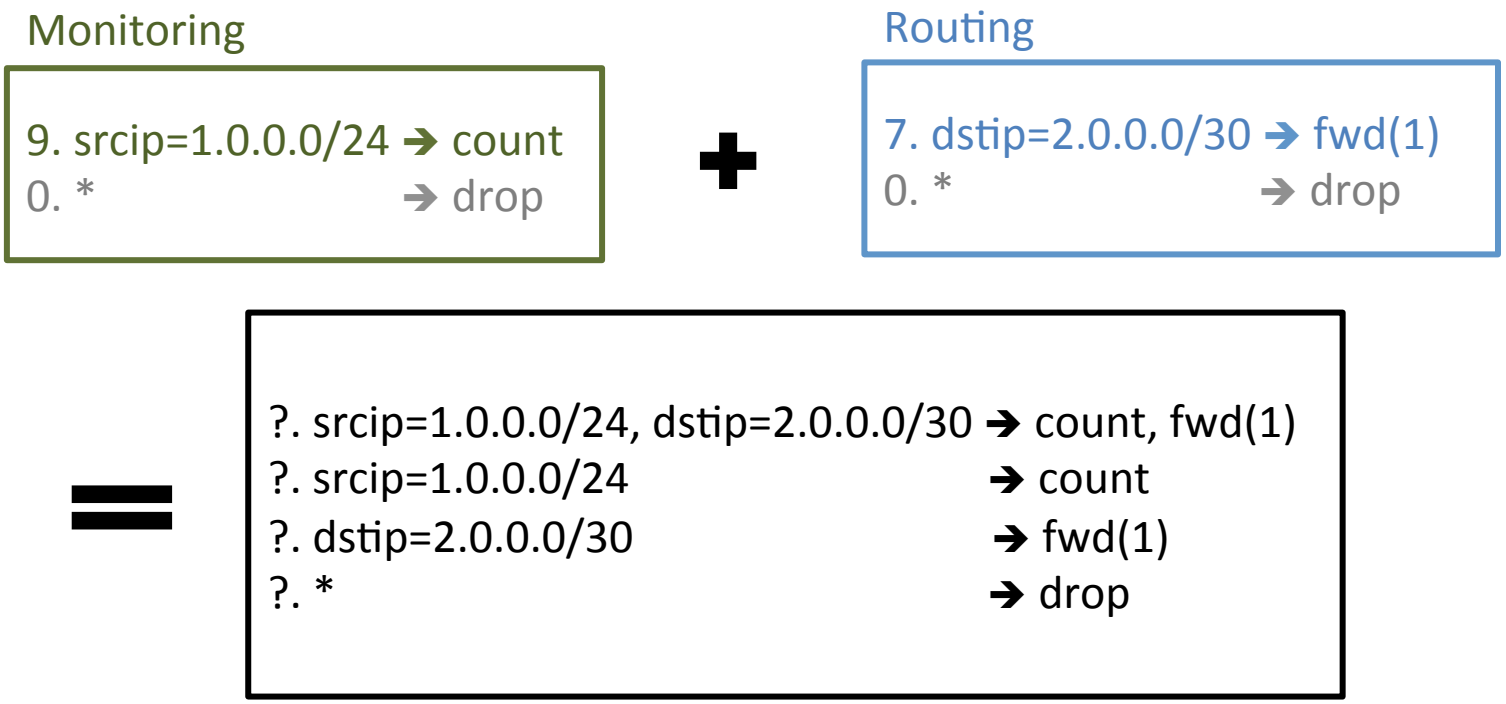
Policy Compilation

- Policy: a list of rules
- Each controller outputs a policy
- Hypervisor compiles them to a single policy



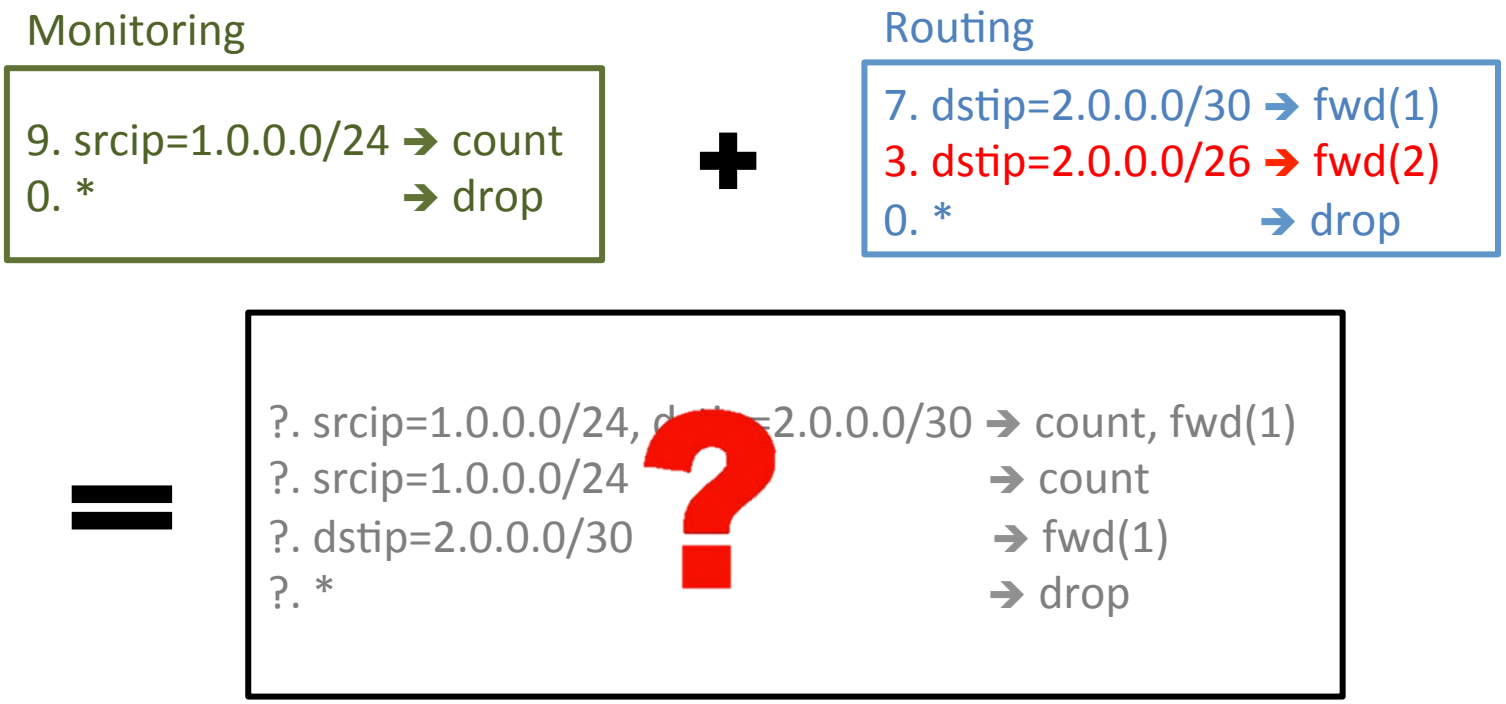
Policy Compilation

- Policy: a list of rules
- Each controller outputs a policy
- Hypervisor compiles them to a single policy



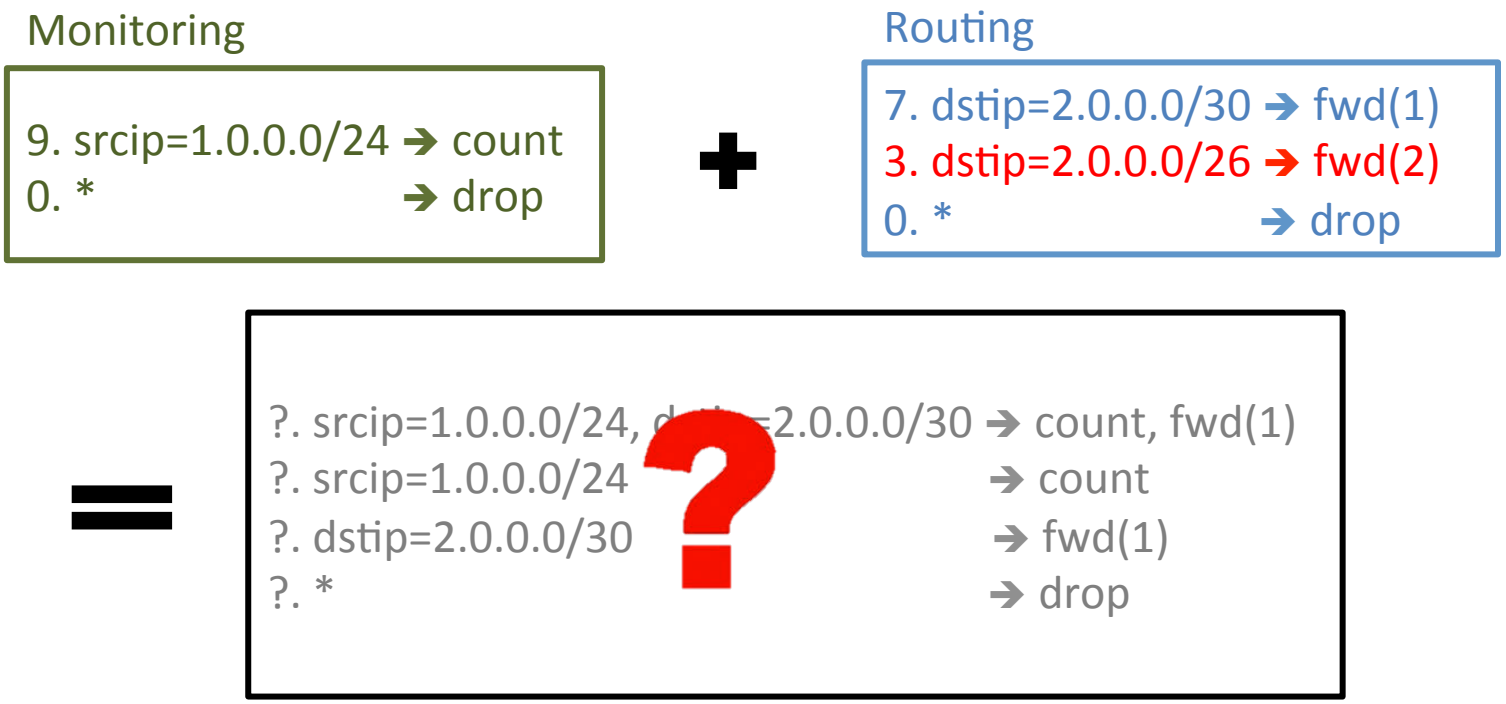
Key challenge: Efficient data plane update

- Controllers continuously update their policies
- Hypervisor recompiles them and update switches



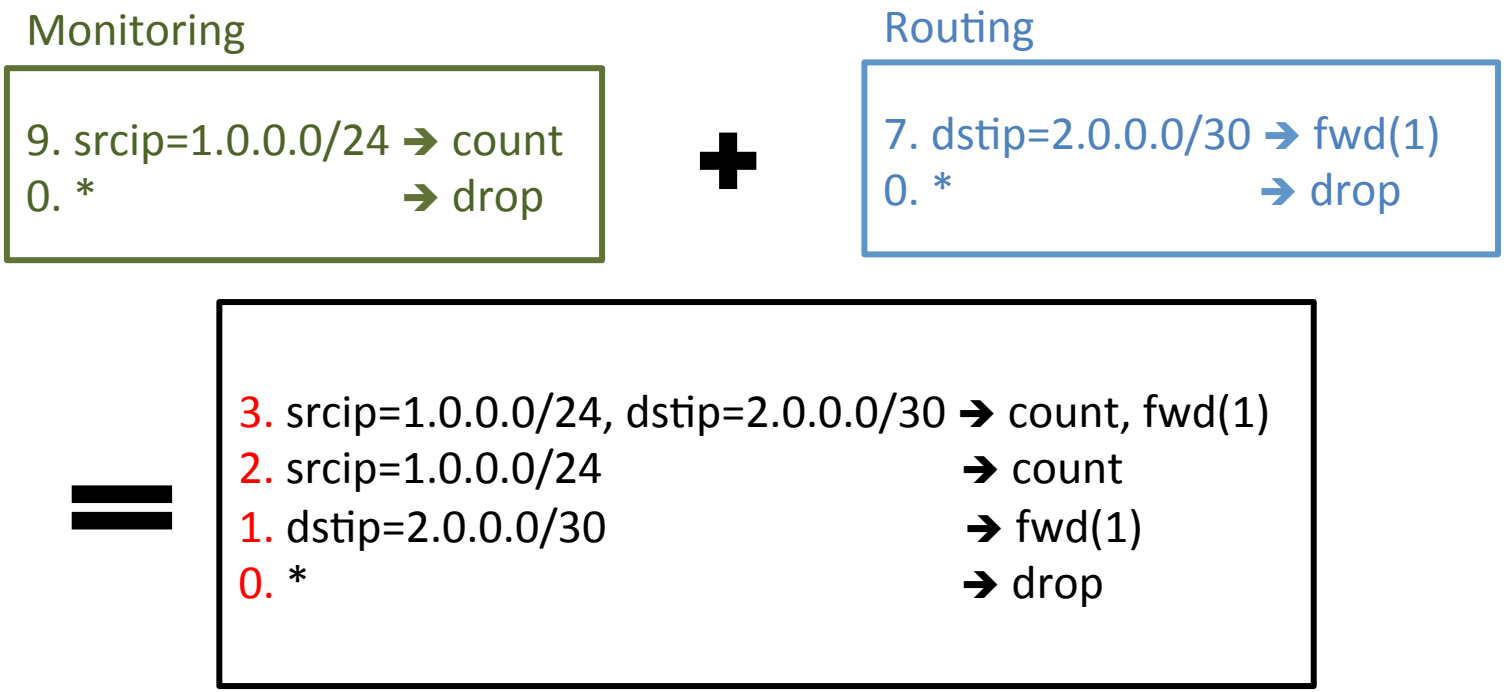
Key challenge: Efficient data plane update

- Computation overhead
  - The computation to recompile the new policy
- Rule-update overhead
  - The rule-updates to update switches to the new policy



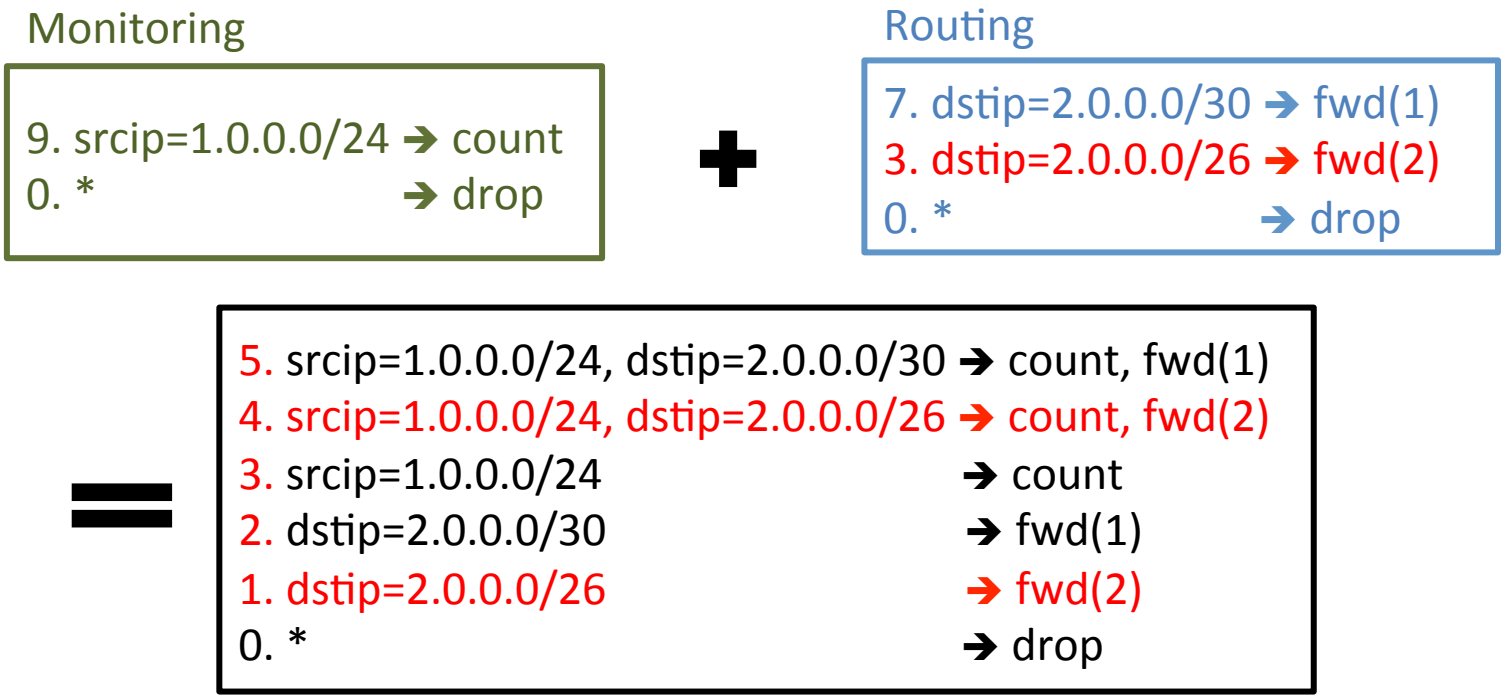
Naïve Solution

- Assign priorities from top to bottom by decrement of 1



Naïve Solution

- Assign priorities from top to bottom by decrement of 1



Naïve Solution

- Assign priorities from top to bottom by decrement of 1

3. srcip=1.0.0.0/24, dstip=2.0.0.0/30 ➔ count, fwd(1)
2. srcip=1.0.0.0/24 ➔ count
1. dstip=2.0.0.0/30 ➔ fwd(1)
0. * ➔ drop

Update

5. srcip=1.0.0.0/24, dstip=2.0.0.0/30 ➔ count, fwd(1)
4. srcip=1.0.0.0/24, dstip=2.0.0.0/26 ➔ count, fwd(2)
3. srcip=1.0.0.0/24 ➔ count
2. dstip=2.0.0.0/30 ➔ fwd(1)
1. dstip=2.0.0.0/26 ➔ fwd(2)
0. * ➔ drop

Computation overhead

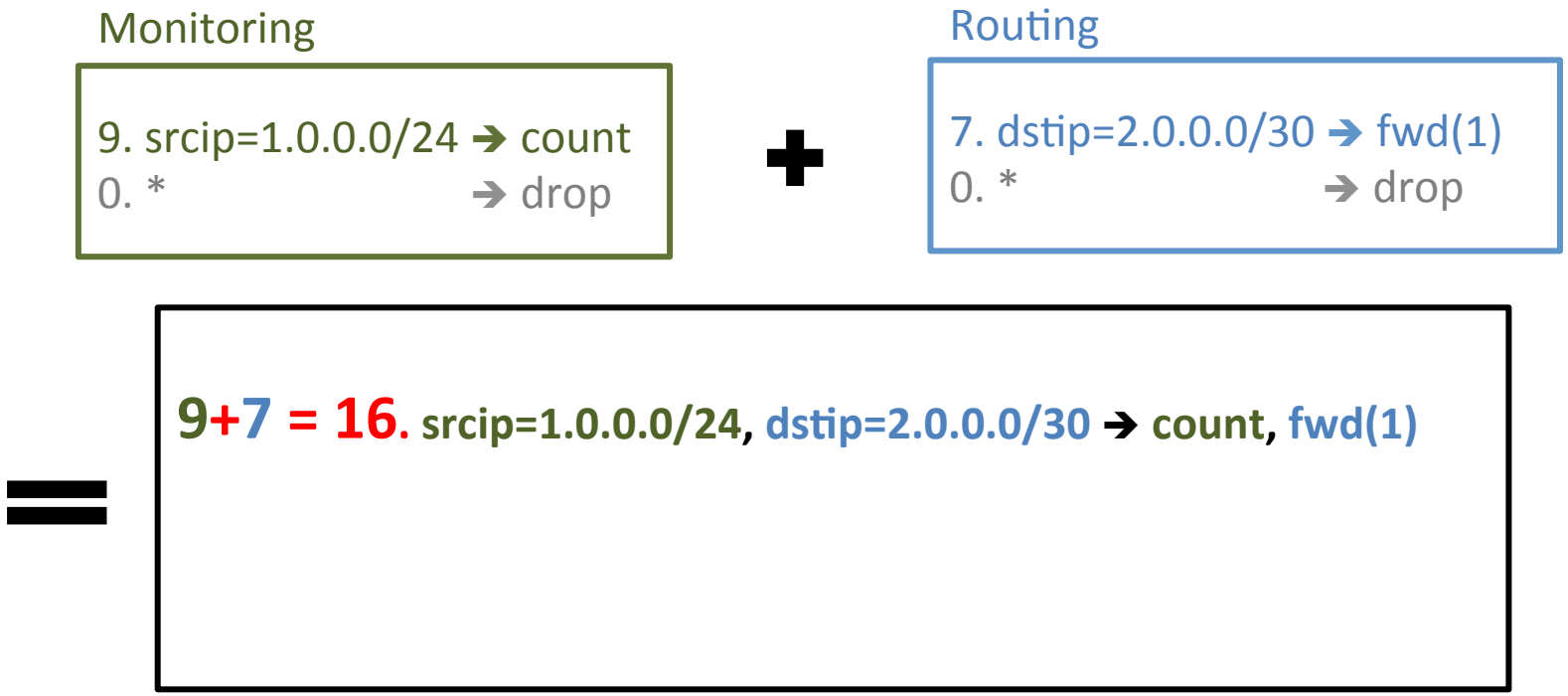
- Recompute the whole switch table and assign priorities

Rule-update overhead

- Only 2 new rules, but 3 more rules change priority

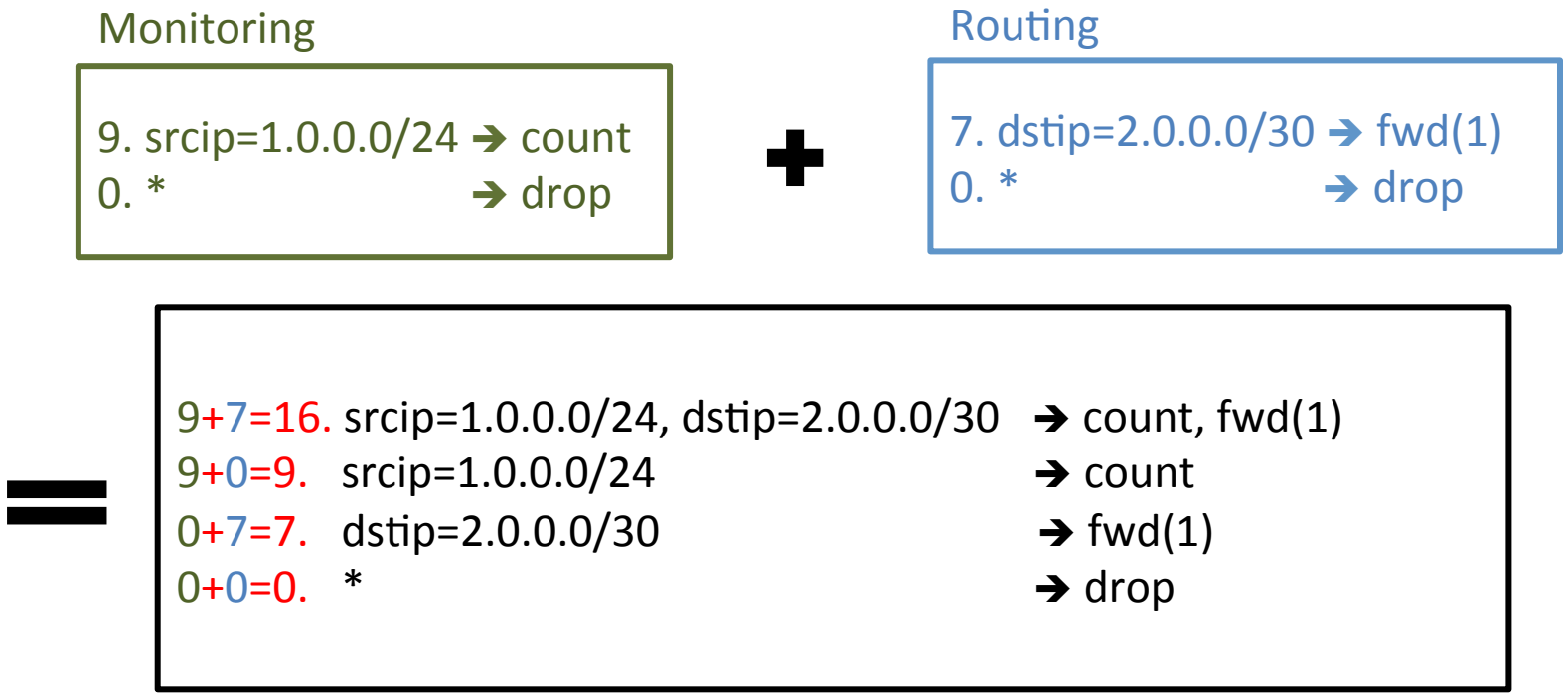
Incremental Update

- Add priorities for parallel composition



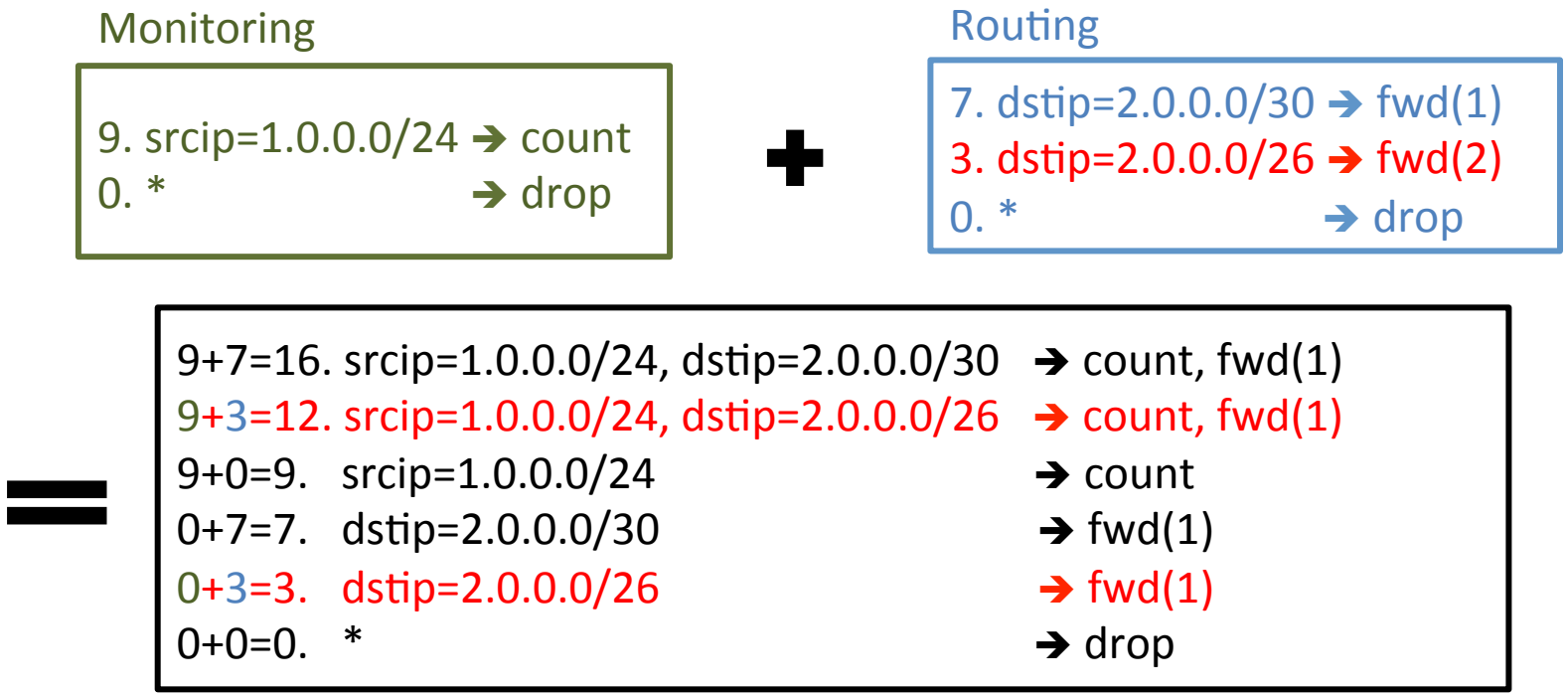
Incremental Update

- Add priorities for parallel composition



Incremental Update

- Add priorities for parallel composition



Incremental Update

- Add priorities for parallel composition

16.	srcip=1.0.0.0/24, dstip=2.0.0.0/30	➔ count, fwd(1)
9.	srcip=1.0.0.0/24	➔ count
7.	dstip=2.0.0.0/30	➔ fwd(1)
0.	*	➔ drop

16.	srcip=1.0.0.0/24, dstip=2.0.0.0/30	➔ count, fwd(1)
12.	srcip=1.0.0.0/24, dstip=2.0.0.0/26	➔ count, fwd(2)
9.	srcip=1.0.0.0/24	➔ count
7.	dstip=2.0.0.0/30	➔ fwd(1)
3.	dstip=2.0.0.0/26	➔ fwd(2)
0.	*	➔ drop

Computation overhead

- Only compose the new rule with rules in monitoring

Rule-update overhead

- Add 2 new rules

Incremental Update

- Add priorities for parallel composition
- Concatenate priorities for sequential composition

Load Balancing

3. srcip=0.0.0.0/2, dstip=3.0.0.0 ➔ dstip=2.0.0.1
1. dstip=3.0.0.0 ➔ dstip=2.0.0.2
0. * ➔ drop

»

Routing

1. dstip=2.0.0.1 ➔ fwd(1)
1. dstip=2.0.0.2 ➔ fwd(2)
0. * ➔ drop

=

.
3 >> 1 = 25, srcip=0.0.0.0/2, dstip=3.0.0.0 ➔ dstip=2.0.0.1, **fwd(1)**
.
.

011	001
High Bits	Low Bits

Incremental Update

- Add priorities for parallel composition
- Concatenate priorities for sequential composition

Load Balancing

```
3. srcip=0.0.0.0/2, dstip=3.0.0.0 ➔ dstip=2.0.0.1
1. dstip=3.0.0.0                  ➔ dstip=2.0.0.2
0. *                              ➔ drop
```

Routing

```
1. dstip=2.0.0.1 ➔ fwd(1)
1. dstip=2.0.0.2 ➔ fwd(2)
0. *             ➔ drop
```

```
25. srcip=0.0.0.0/2, dstip=3.0.0.0 ➔ dstip=2.0.0.1, fwd(1)
9. dstip=3.0.0.0                   ➔ dstip=2.0.0.2, fwd(2)
0. *                               ➔ drop
```

Incremental Update

- Add priorities for parallel composition
- Concatenate priorities for sequential composition

Load Balancing

3. srcip=0.0.0.0/2, dstip=3.0.0.0 ➔ dstip=2.0.0.1
2. srcip=0.0.0.0/1, dstip=3.0.0.0 ➔ dstip=2.0.0.3
1. dstip=3.0.0.0 ➔ dstip=2.0.0.2
0. * ➔ drop

»

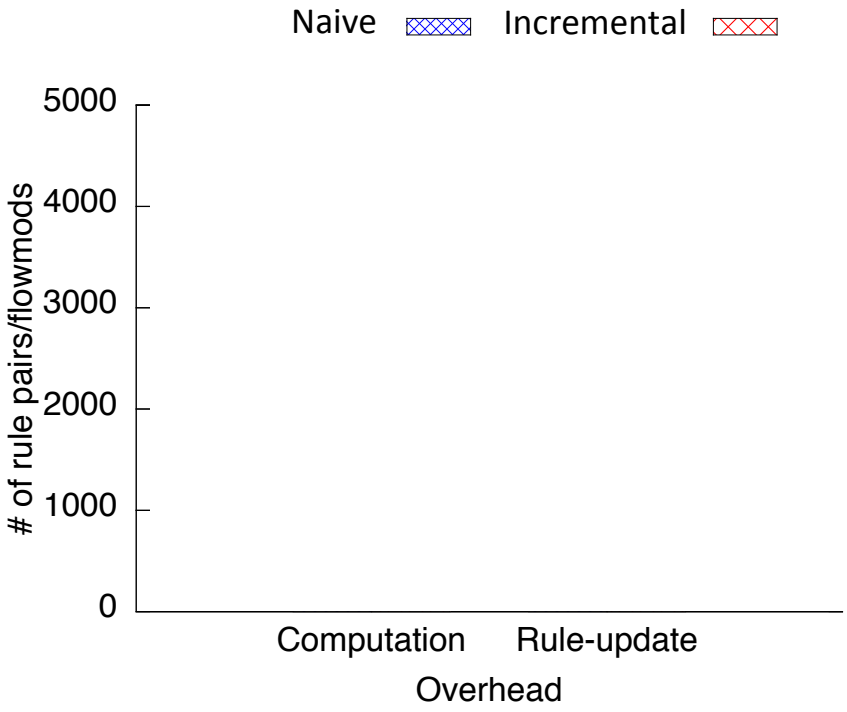
Routing

1. dstip=2.0.0.1 ➔ fwd(1)
1. dstip=2.0.0.2 ➔ fwd(2)
1. dstip=2.0.0.3 ➔ fwd(3)
0. * ➔ drop

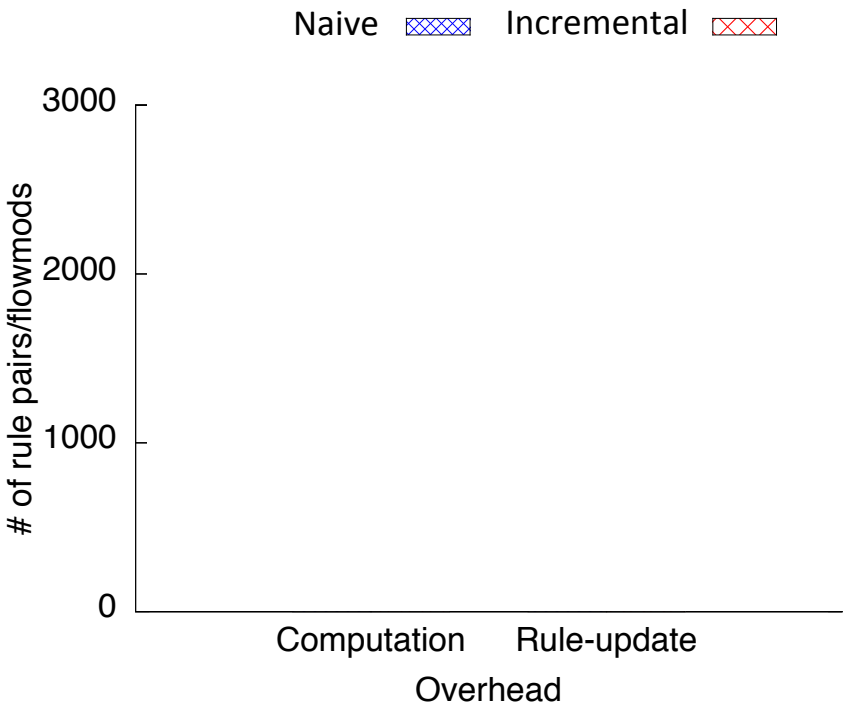
=

25. srcip=0.0.0.0/2, dstip=3.0.0.0 ➔ dstip=2.0.0.1, fwd(1)
17. srcip=0.0.0.0/1, dstip=3.0.0.0 ➔ dstip=2.0.0.3, fwd(3)
9. dstip=3.0.0.0 ➔ dstip=2.0.0.2, fwd(2)
0. * ➔ drop

Evaluation

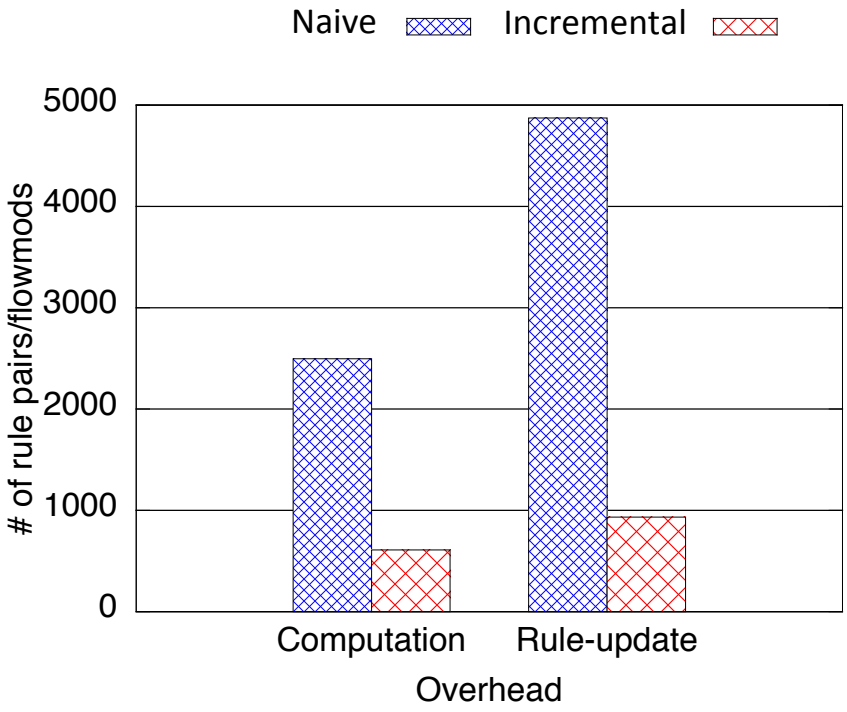


Monitoring + Routing

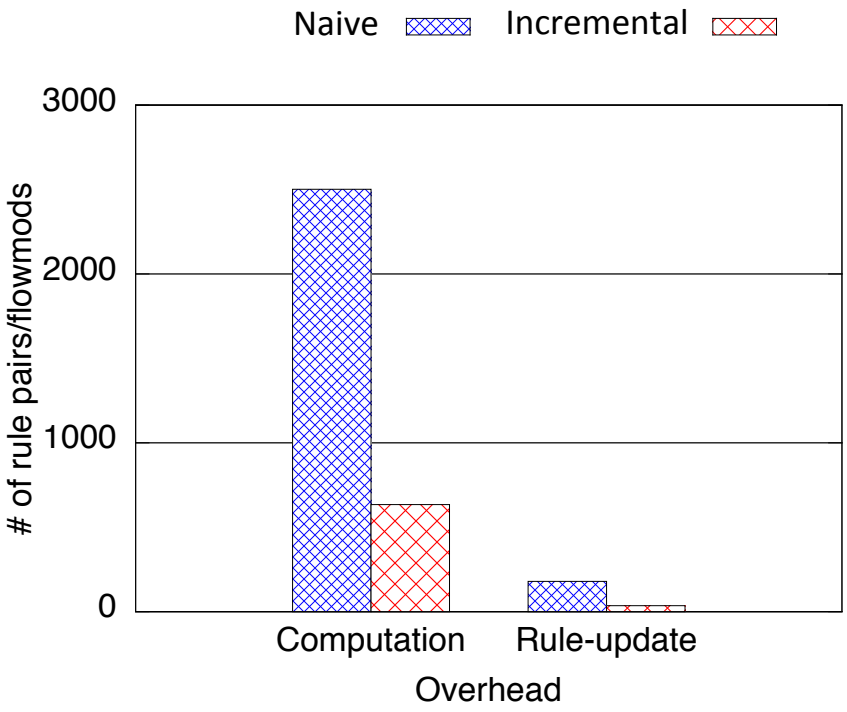


Load balancing >> Routing

Evaluation



Monitoring + Routing



Load balancing >> Routing

Reduce computation overhead by 4x, rule updates by 5x

Conclusion

- Compositional network hypervisor
- Novel algorithm to efficiently update the data plane
- Ongoing work: prototype in OpenVirteX

Thanks!