Use cases of Programmable Dataplane (P4)

ECE/CS598HPN

Radhika Mittal

Which paper(s) did you read?

- (A) BeauCoup: Network Monitoring
- (B) Elmo: Multicast
- (C) Both
- (D) Neither

Network Monitoring

- Most popular usecase of programmable dataplanes.
- Lots of recent papers!
- Key challenges:
  - Dealing with small amount of memory.
  - Ensuring high line rate (small processing capability, limited memory access)
  - Supporting a wide variety of queries.

[1][bo’ku] *Adv*. many, a lot.

BeauCoup:[1]

Answering *many* network traffic queries, *one* memory update at a time!

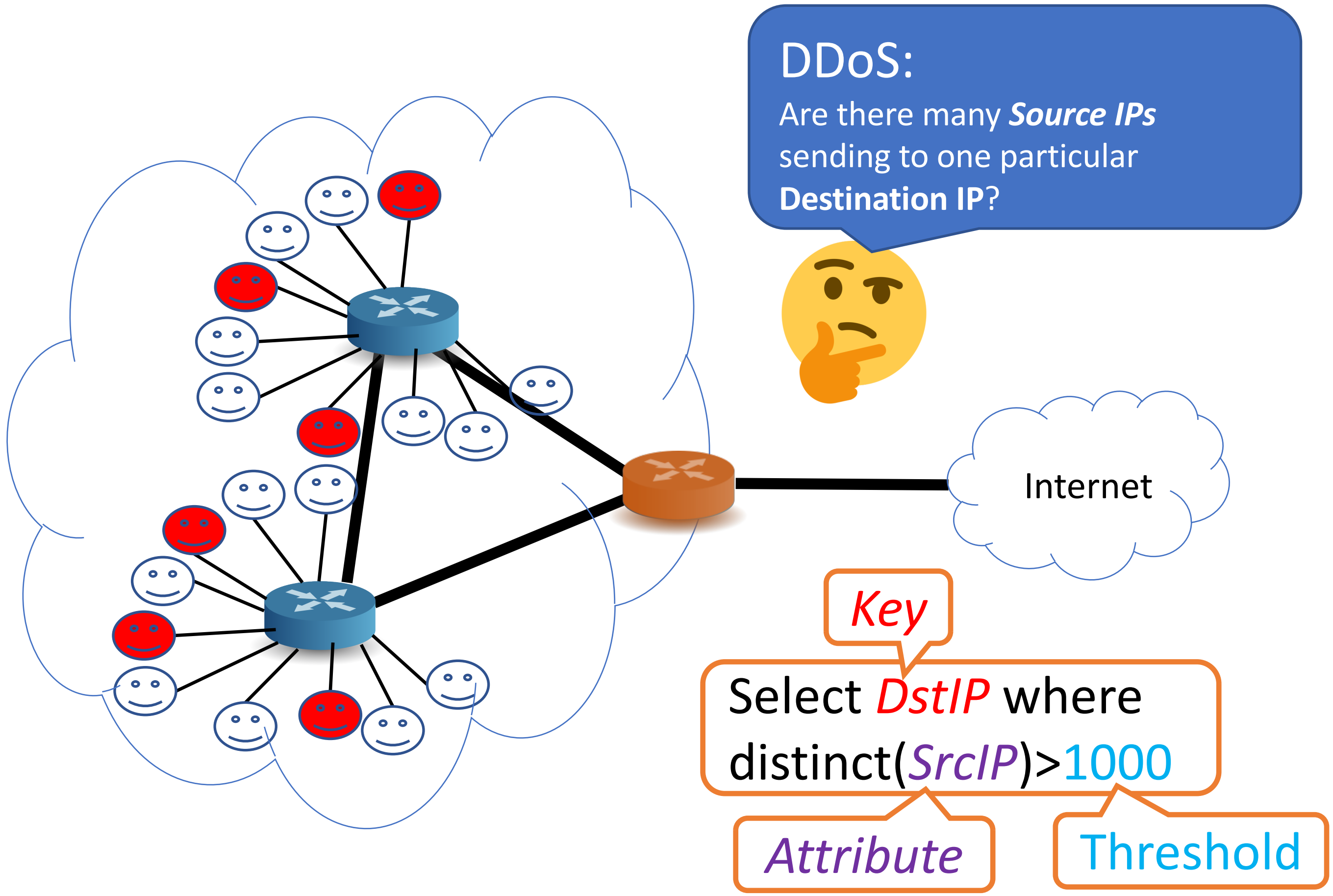
Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, Jennifer Rexford

SIGCOMM 2020 new york city USA

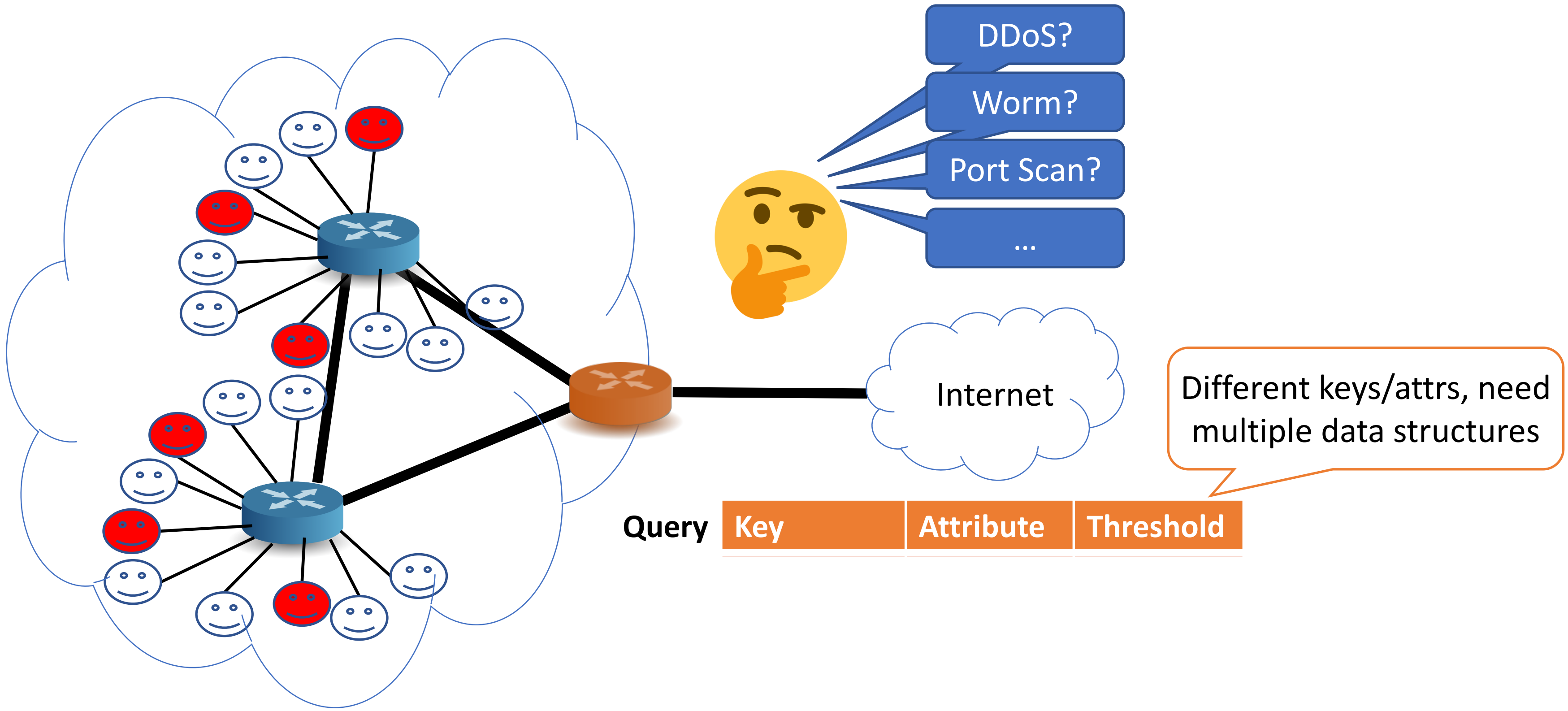
PRINCETON UNIVERSITY

Slides from Xiaoqi Chen

Network traffic query



Many network traffic queries



Many network traffic queries

I have 42 queries

Run 42 data structures?

I can't…

Spec for today's commodity programmable switch:

- XX Tbps aggregated throughput
- YY MB data-plane memory
- **Can only access ZZ bytes of memory per packet**

One memory update at a time?

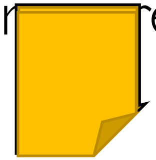
- Constant memory update per packet, regardless of the number of queries?
- Game plan:
  1. Each query uses only **o(1)** memory update per packet ***on average***
  2. Combine many different queries, on average uses **O(1)**
  3. Coordinate, ***at most*** **O(1)** per packet

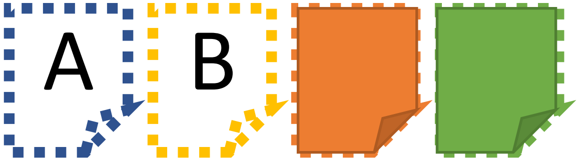
BeauCoup's Approach

- Challenge:
  many queries, few memory updates
- Achieving **o(1)** memory access:
  coupon collectors

The coupon collector problem

- 4 different coupons, collect all of them
- Random draws
- How many total draws are required?





Naïve Approach

Query: Select *DstIP* where distinct(*SrcIP*)>130

- Map each ScrIP to a coupon
  - How many total coupons?
  - How many do you need to collect?

- Issues with this approach:
  - Too much memory
  - Each packet results in a coupon collection.
    - Exceed O(1) access when multiple such queries are combined.

BeauCoup coupon collector

f(*SrcIP*) -> Coupon ?

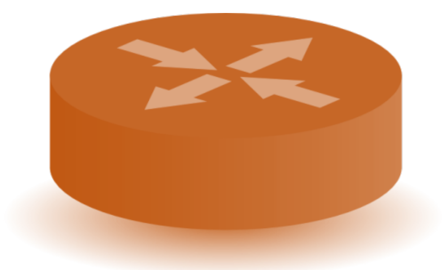
Mapping

Select *DstIP* where distinct(*SrcIP*)>100

Collect different coupons

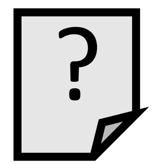
Key: 162.249.4.107

Coupons: A D



- ***f***(10.0.1.15) -> Coupon C
- ***f***(10.0.1.33) -> Coupon B
- ***f***(10.0.1.15) -> Coupon C
- ***f***(10.0.1.42) -> No Coupon

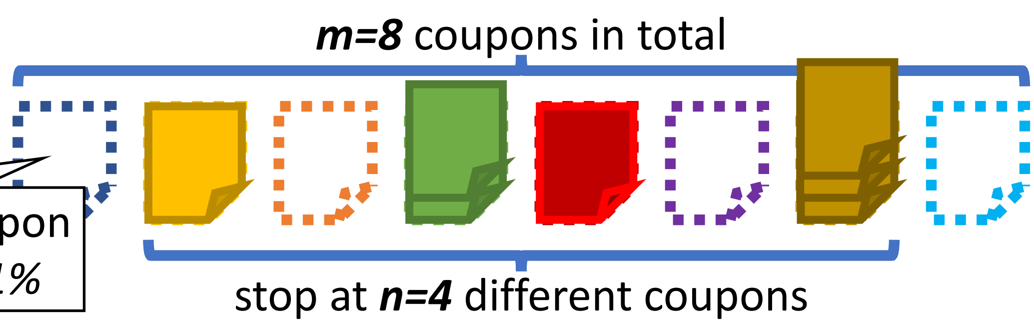
BeauCoup coupon collector

f(*SrcIP*) -> Coupon 

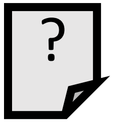
- Generalization: (***m***, ***p***, ***n***)-coupon collector
- ***m*p***<1, most packets collect no coupon

Example:
(***m***=8, ***p***=1%,
n=4)

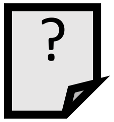
Given a new SrcIP, each coupon is drawn with probability *1%*



Stacking queries: same attribute

q_1: f(*SrcIP*) -> Coupon 

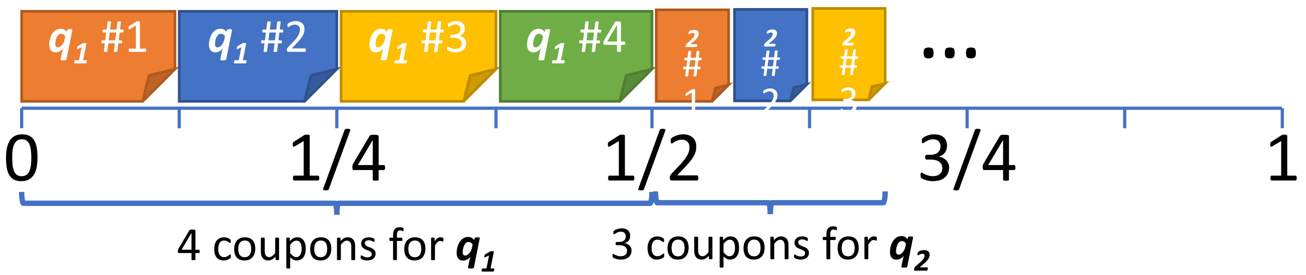
m_1=4, p_1=1/8

q_2: f(*SrcIP*) -> Coupon 

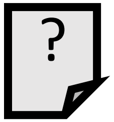
m_2=3, p_2=1/16

⋮

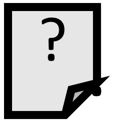
Hash function
h_1(*SrcIP*) -> [0,1)



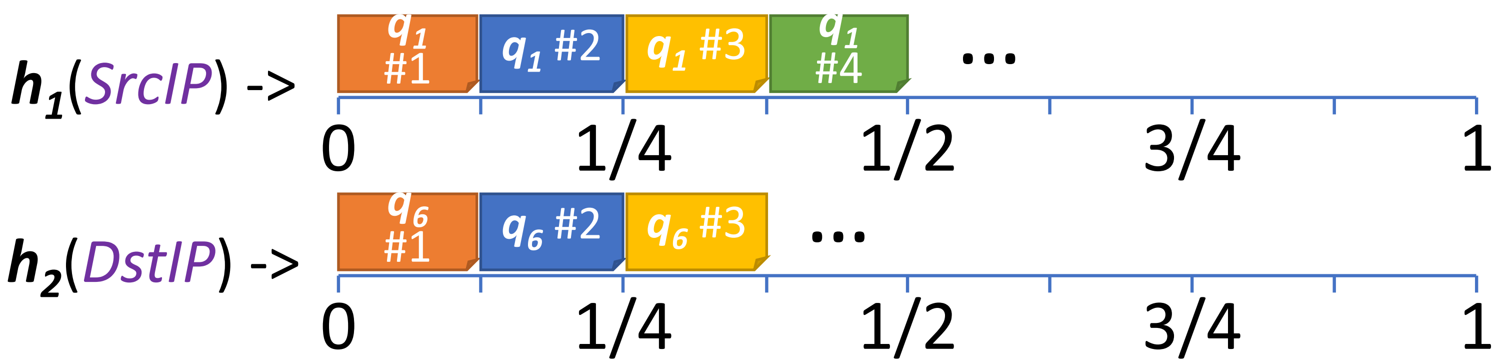
One hash function for each attribute

q_1: f(*SrcIP*) -> Coupon 

m_1=4, p_1=1/8

q_6: g(*DstIP*) -> Coupon 

m_6=3, p_6=1/8

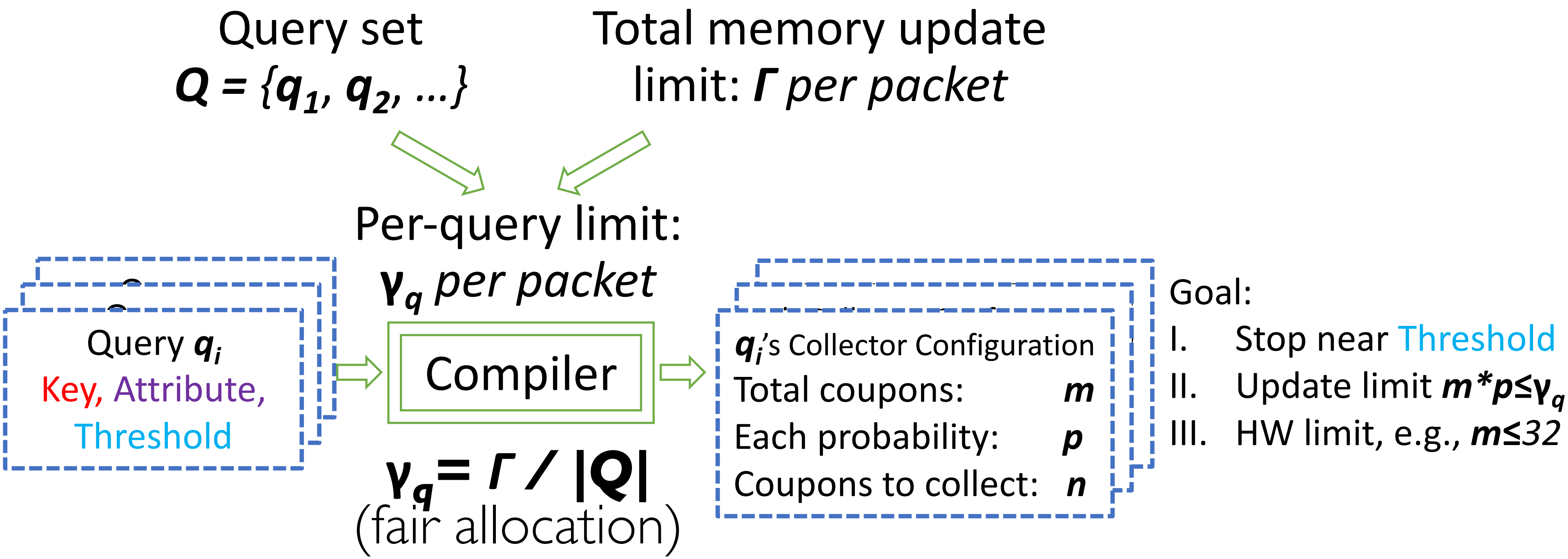


Randomly break ties if a coupon needs to be collected for two different attributes

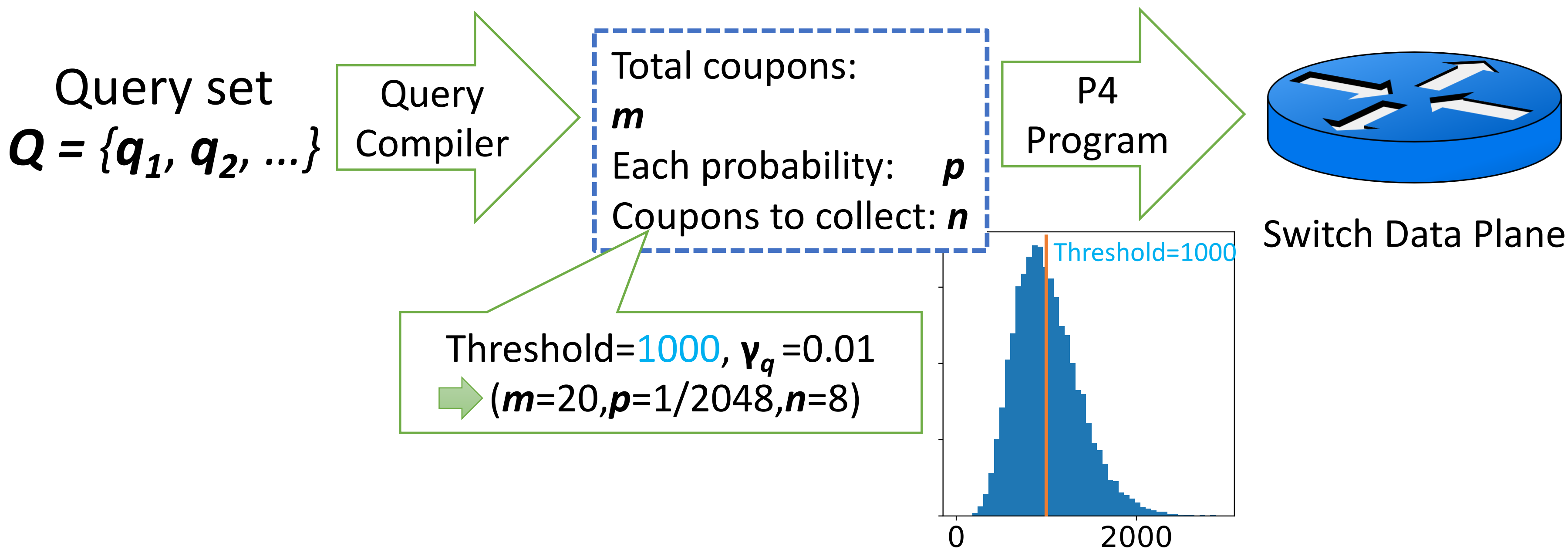
System design

- Query compiler: finds coupon collector configurations
  - Stops near query thresholds, minimize error
  - Hardware limits (e.g., memory access limit)
  - Fairness across queries
- Data plane program: collect coupons into in-memory table
  - Simultaneously run ***many*** queries
  - At most ***one*** coupon per packet
  - Update queries on-the-fly

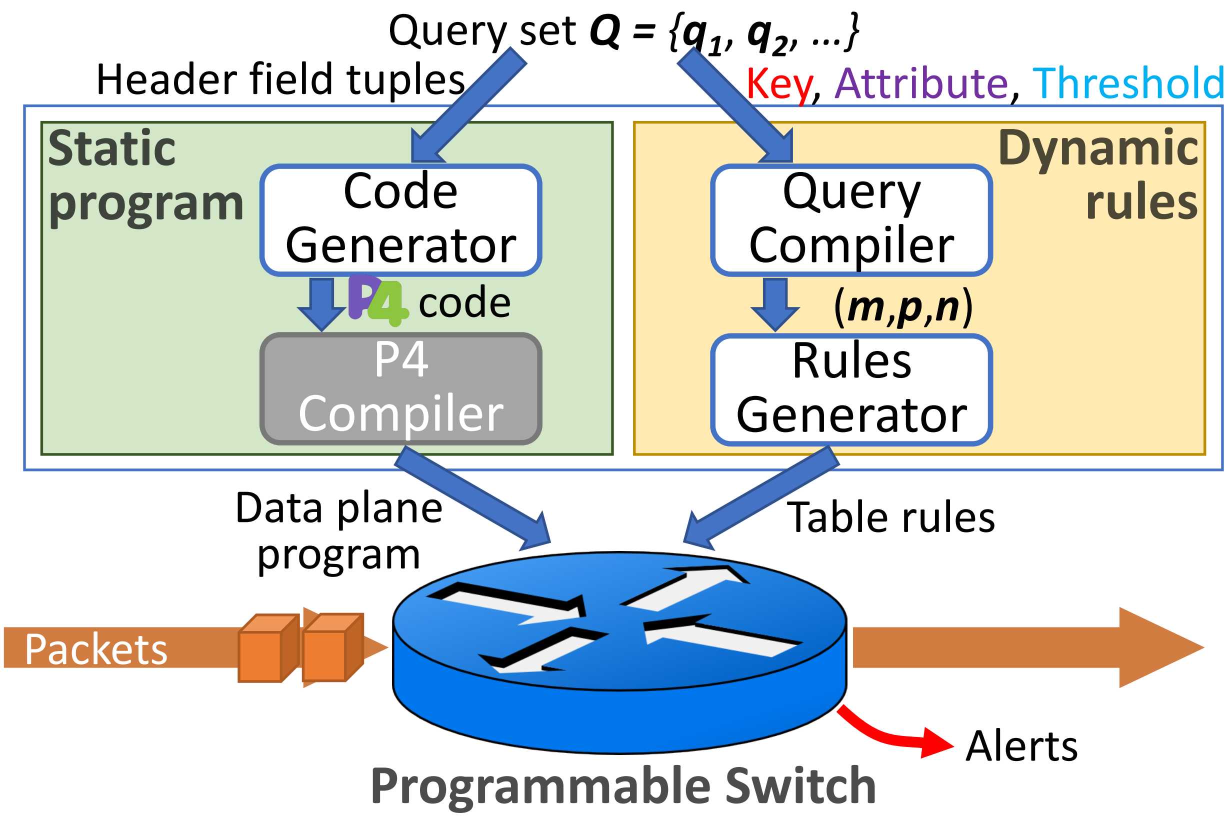
Query compiler



Query compiler

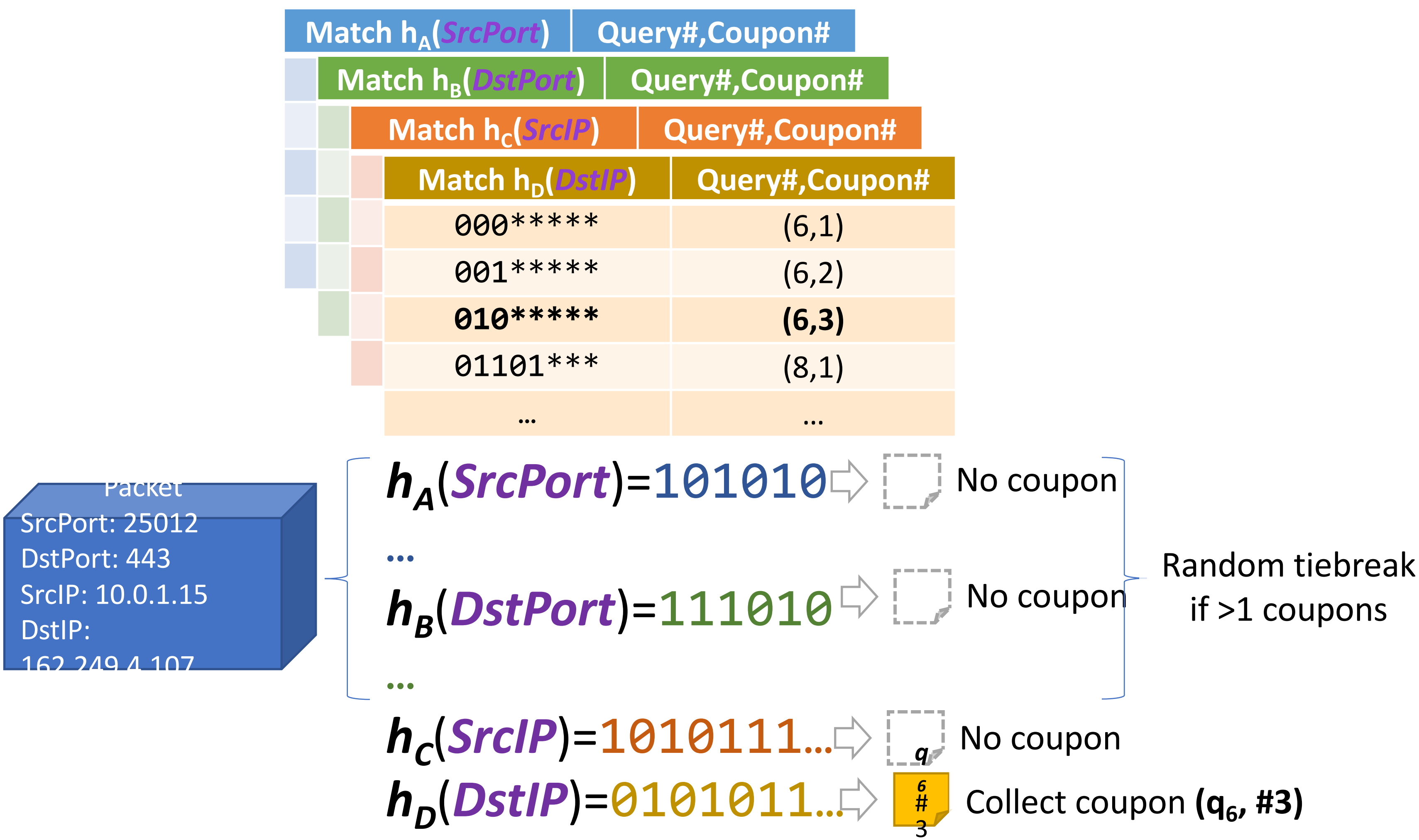


Installing queries into switches

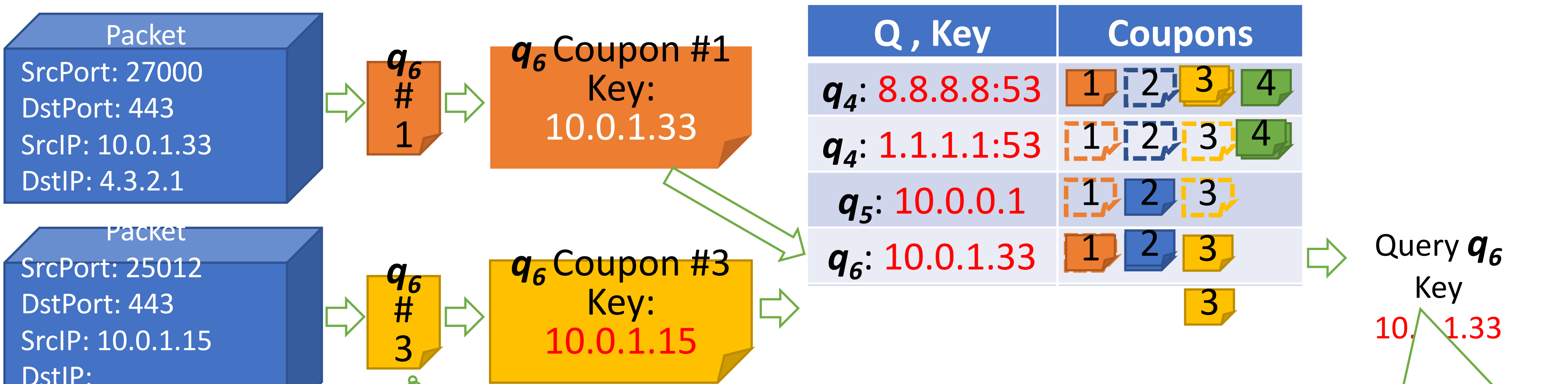


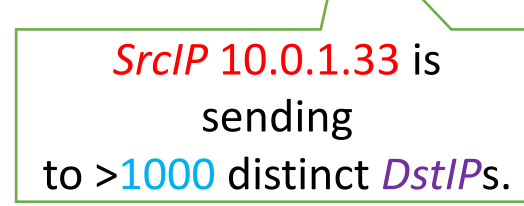
- The installed rules represent query set ***Q***
- Update queries ***on the fly***, without recompiling P4

TCAM for selecting a coupon



Coupon collector table in SRAM





Space efficiency:

- Keys from all queries multiplexed into one table
- Only keep rows for “active keys” (at least one coupon)
- Clear rows after timeout

Evaluation highlights

- How efficient is BeauCoup?

Uses **4x~10x fewer memory access** than the state-of-the-art to achieve the same accuracy.

- How much hardware resource?

On the Barefoot Tofino programmable switch, BeauCoup occupies **<50% of each resource**

BeauCoup:

Answering ***many*** network traffic queries, ***one*** memory update at a time!

- **Scalable**: built upon *coupon collectors*, runs many queries simultaneously
- **Versatile**: change queries on the fly, without recompiling P4 program
- **Efficient**: achieve the same accuracy using 4x-10x fewer memory accesses

Is this a good usecase of programmable dataplanes?

What are the limitations?

Elmo: Source Routed Multicast for Public Clouds

Muhammad Shahbaz

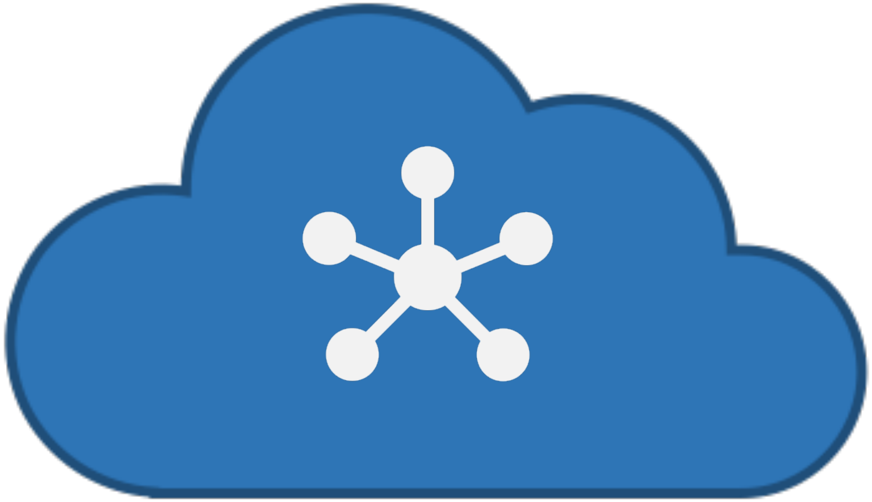
Lalith Suresh, Jennifer Rexford, Nick Feamster,

Ori Rottenstreich, and Mukesh Hira

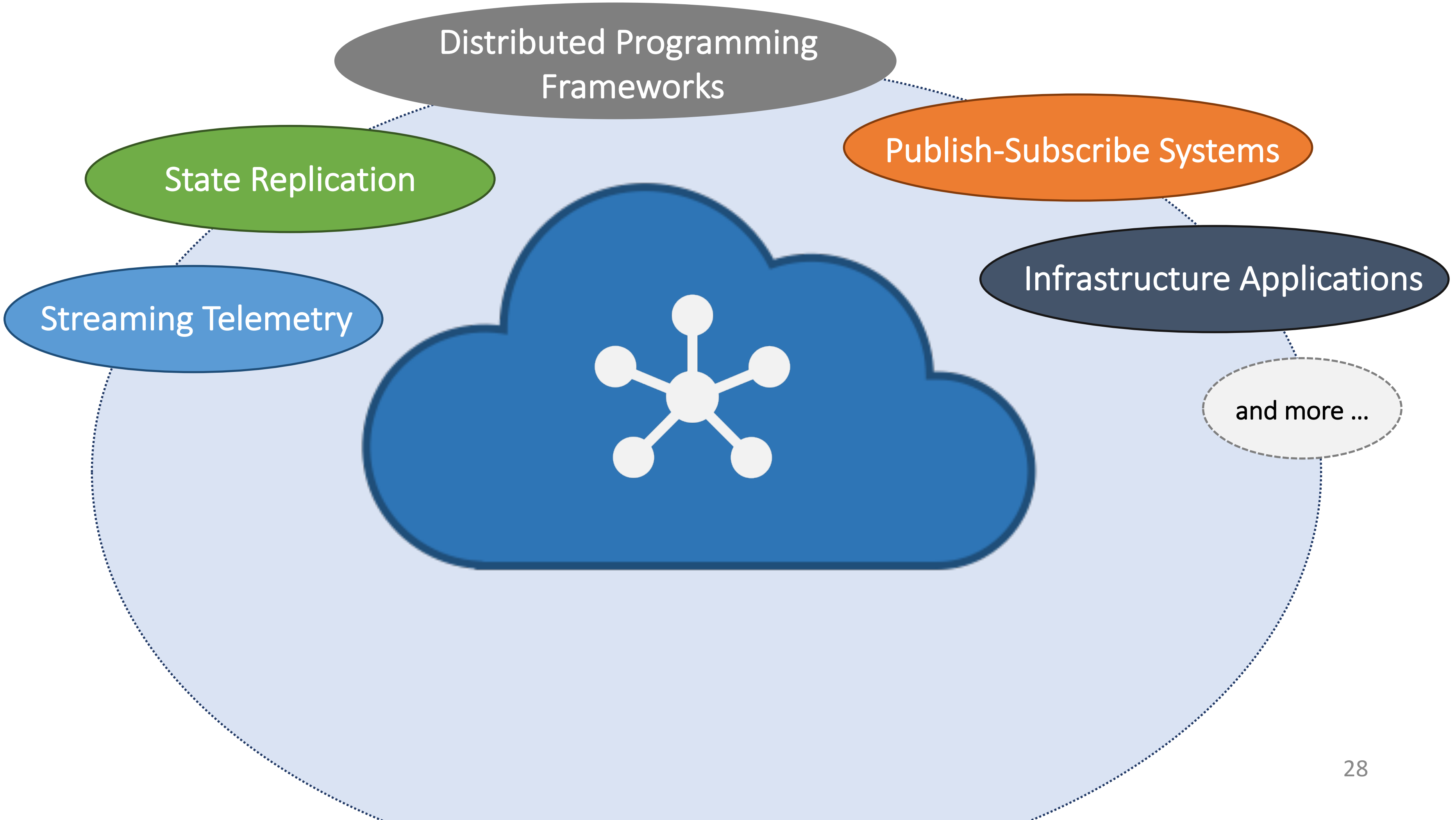
Stanford University PRINCETON UNIVERSITY Technion Israel Institute of Technology vmware

Slides from Muhammad Shahbaz

1-to-Many Communication in Cloud



1-to-Many Communication in Cloud

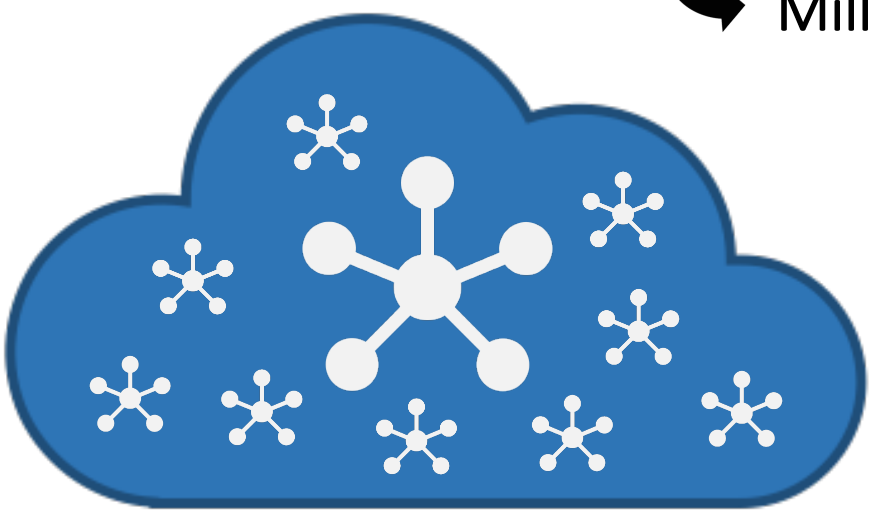


1-to-Many Communication in Cloud

10,000s of tenants

→ 100s of workloads

→ Millions of groups



amazon Google Microsoft

1-to-Many Communication in Cloud

10,000s of tenants

→ 100s of workloads

→ Millions of groups

amazon Google Microsoft

1-to-Many Communication in Cloud

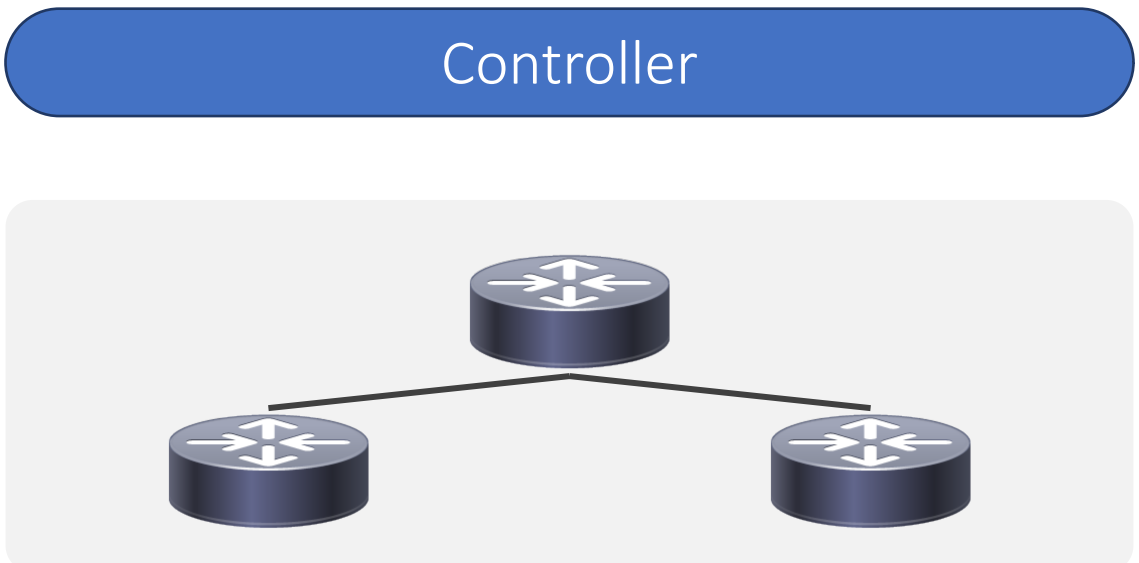
10,000s of tenants

→ 100s of workloads

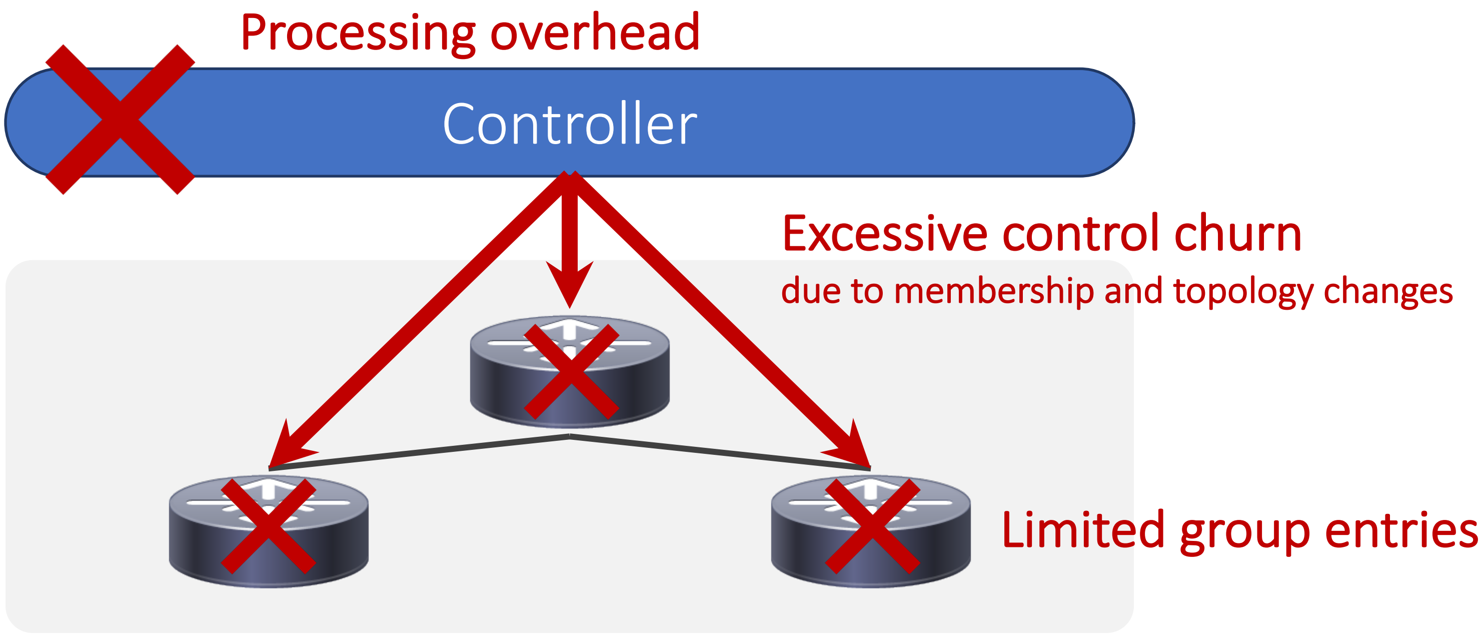
→ Millions of groups

amazon Google Microsoft

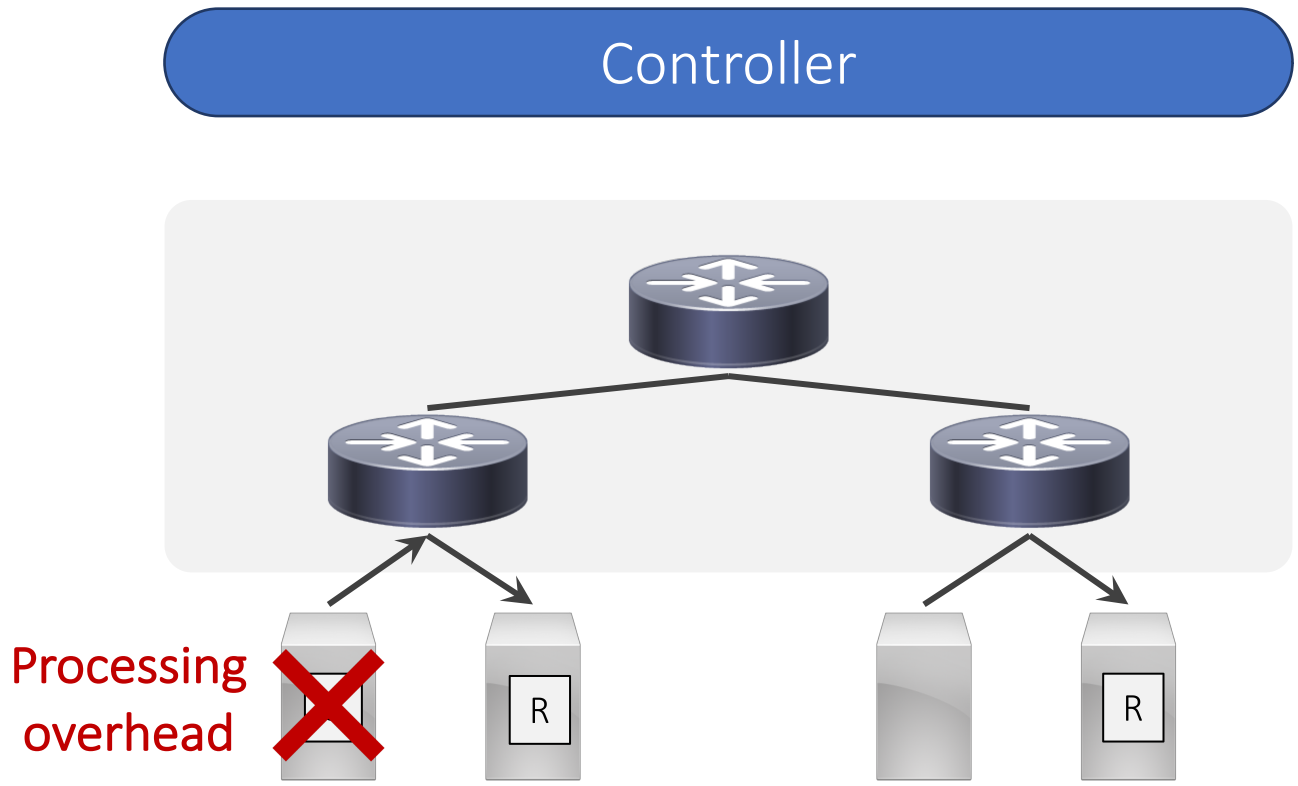
Limitations of Native Multicast



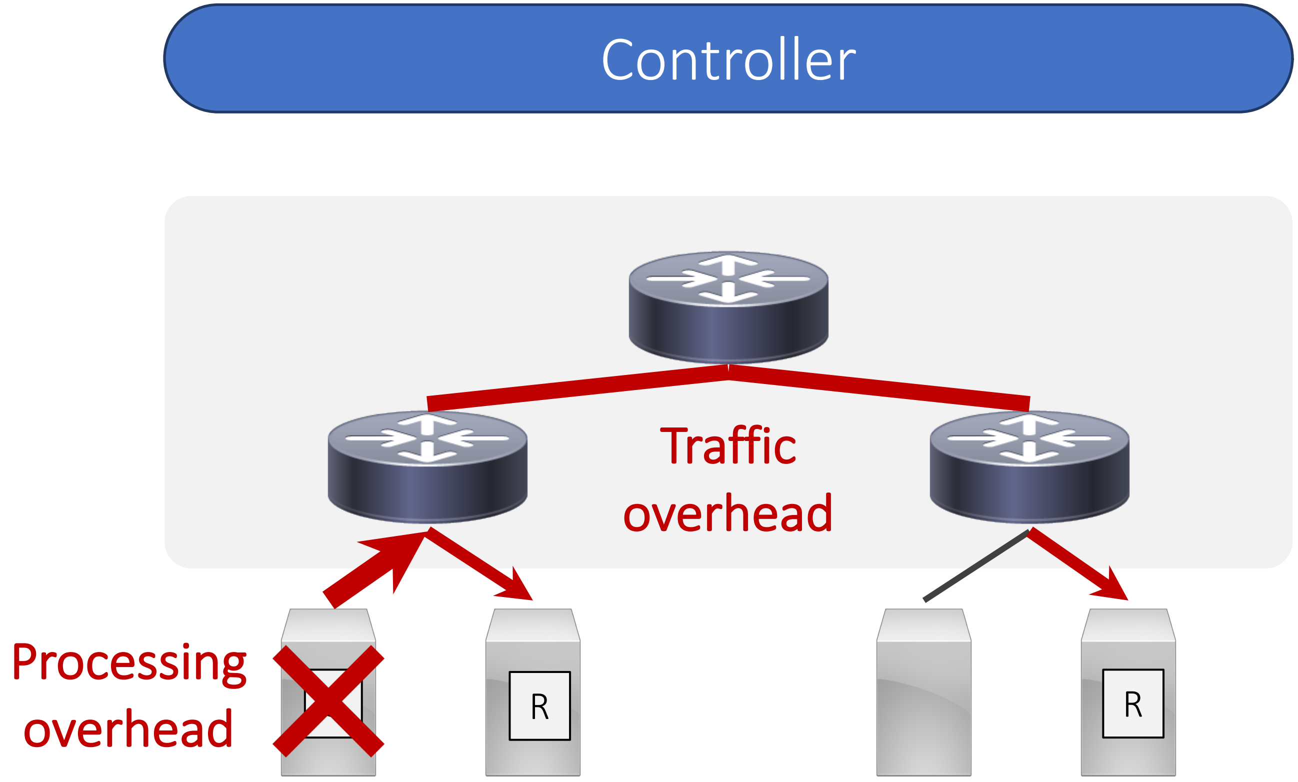
Limitations of <u>Native</u> Multicast



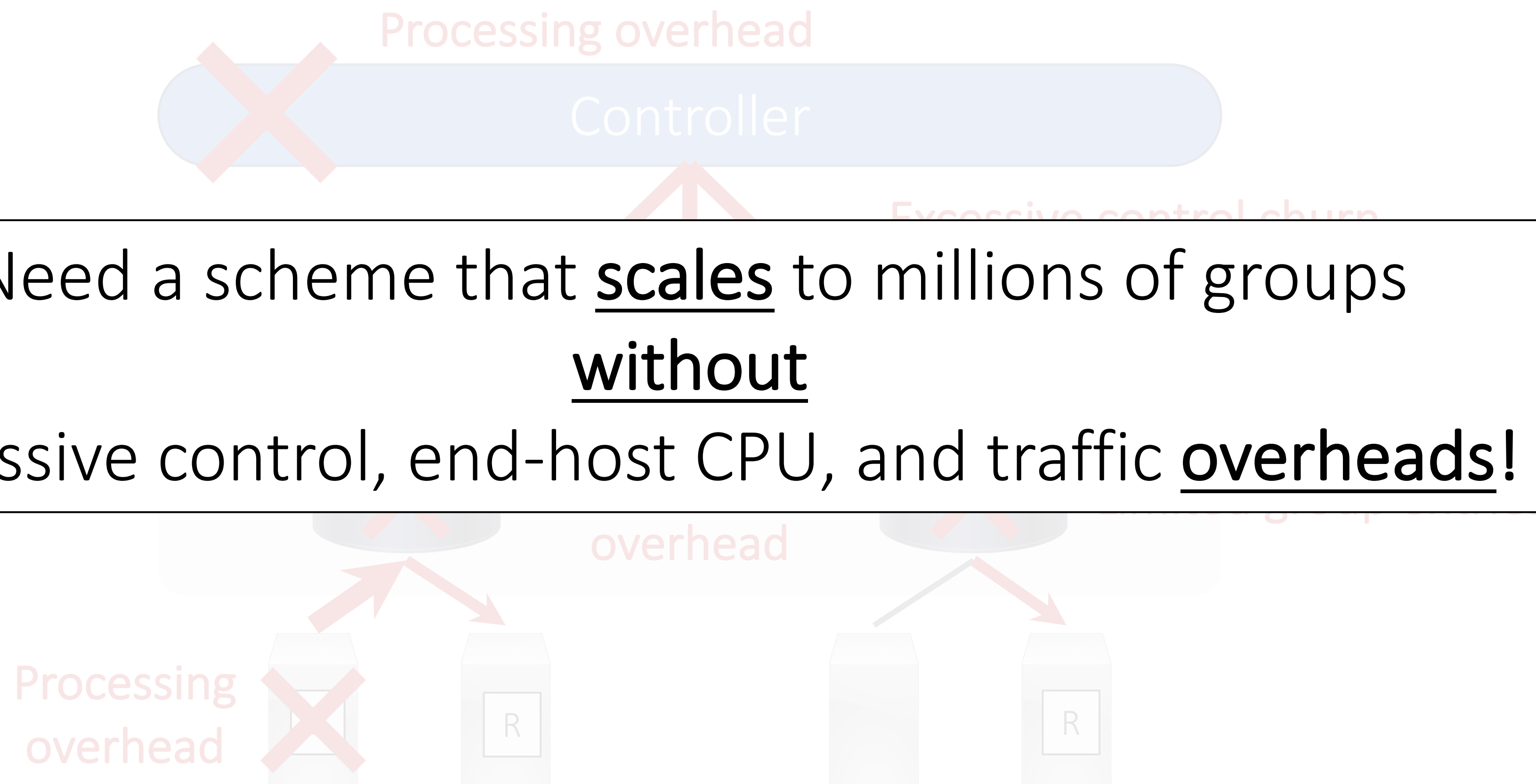
Restricted to Unicast-based Alternatives



Restricted to Unicast-based Alternatives

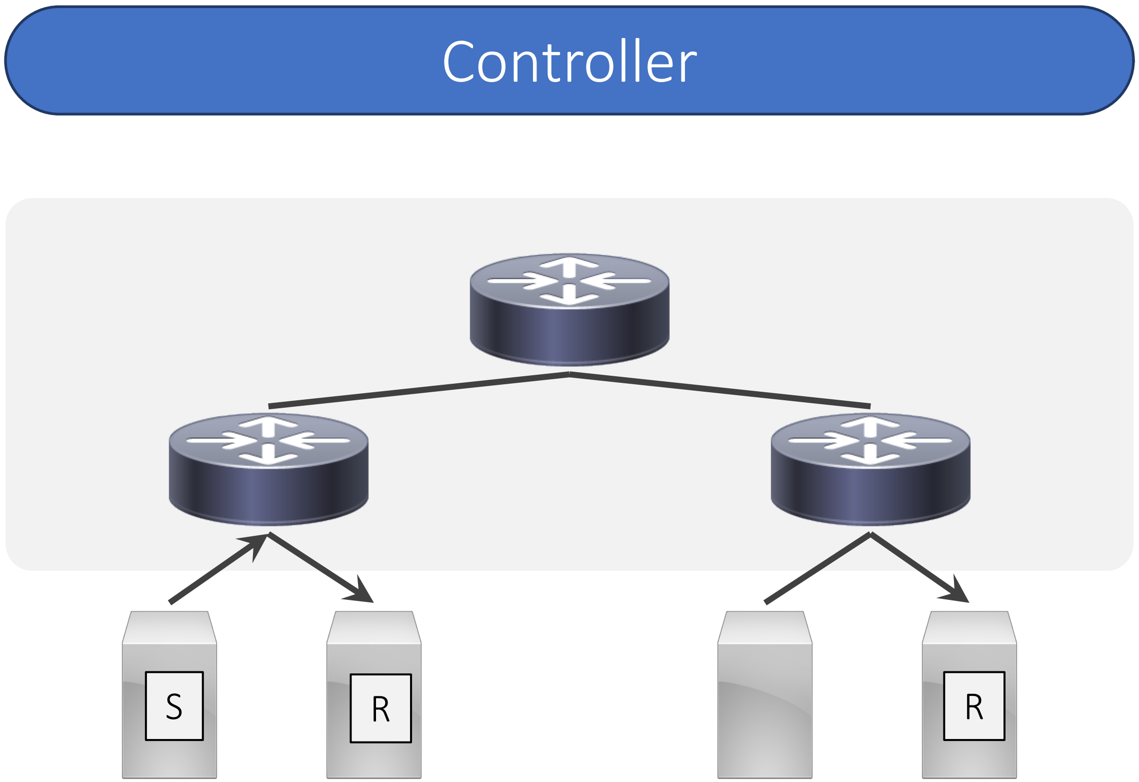


1-to-Many Communication in the Cloud

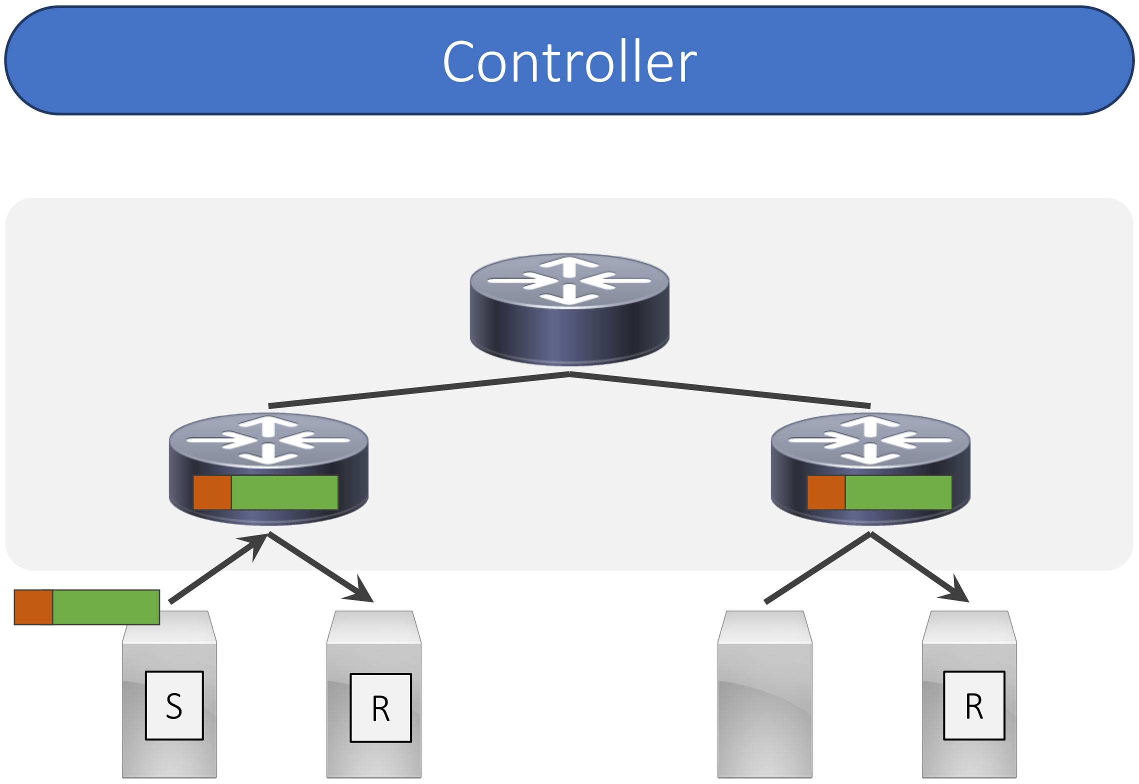


Need a scheme that **scales** to millions of groups
without
excessive control, end-host CPU, and traffic **overheads**!

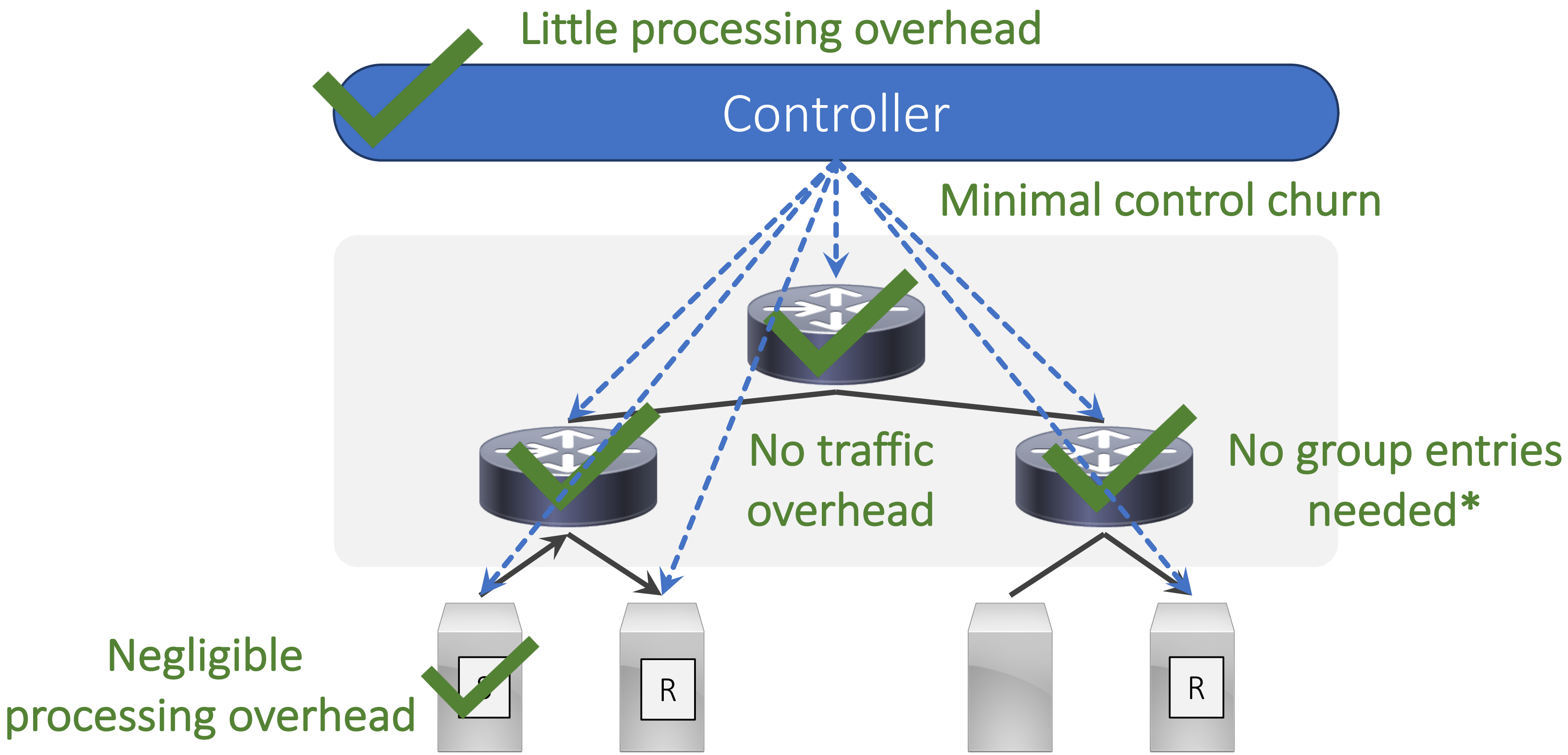
Proposal: Source Routed Multicast



Proposal: Source Routed Multicast



Proposal: Source Routed Multicast



A Naïve Source Routed Multicast

A **multicast group** encoded as a list of **(Switch, Ports)** pairs

```
Switch 1: [Ports ]
Switch 2: [.. .. ..]
Switch 3: [.. .. ..]
Switch 4: [.. .. .x ..]
Switch 5: [.x .. .. ..]
```

20x the packet size!

For a data center with:

- **1000** switches
- **48 ports** per switch

O(30) bytes per switch

O(30,000) bytes header

for a group spanning 1000 switches

Enabling Source Routed Multicast in Public Clouds

Key attributes:

- Efficiently encode multicast forwarding policy inside packets
- Process this encoding at hardware speed in the switches
- Execute tenants' applications without modification

Encoding a Multicast Policy in Elmo

A multicast group encoded as
a list of **(Switch, Ports)** pairs

```
Switch 1: [Ports ]
Switch 2: [.. .. ..]
Switch 3: [.. .. ..]
Switch 4: [.. .. .x ..]
Switch 5: [.x .. .. ..]
```

Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

```
Switch 1: [Bitmap]
Switch 2: [.. .. ..]
Switch 3: [.. .. ..]
Switch 4: [.. .. .x ..]
Switch 5: [.x .. .. ..]
```

1 Encode switch ports as a **bitmap**

Bitmap is the internal **data structure** that switches use for **replicating packets**

Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

```
Switch 1: [Bitmap]
Switch 2: [.. .. ..]
Switch 3: [.. .. ..]
Switch 4: [.. .. .x ..]
Switch 5: [.x .. .. ..]
```

2 Group switches into **layers**

Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

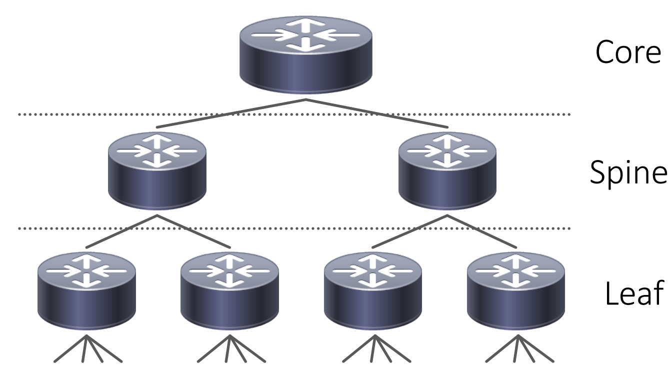
```
Switch 1: [Bitmap]
Switch 2: [.. .. ..]
Switch 3: [.. .. ..]
Switch 4: [.. .. .x ..]
Switch 5: [.x .. .. ..]
```

Core

Spine

Leaf

2 Group switches into **layers**



More precisely: *upstream leaf, upstream spine, core, downstream spine, downstream leaf*

Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

```
Switch 1: [Bitmap]
Switch 2: [.. .. ..]
Switch 3: [.. .. ..]
Switch 4: [.. .. .x ..]
Switch 5: [.x .. .. ..]
```

3 Switches within a layer with **same** ports **share a bitmap**

Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

```
Switch 1: [Bitmap]
Switch 2,3: [.. .. ..]
Switch 4: [.. .. .x ..]
Switch 5: [.x .. .. ..]
```

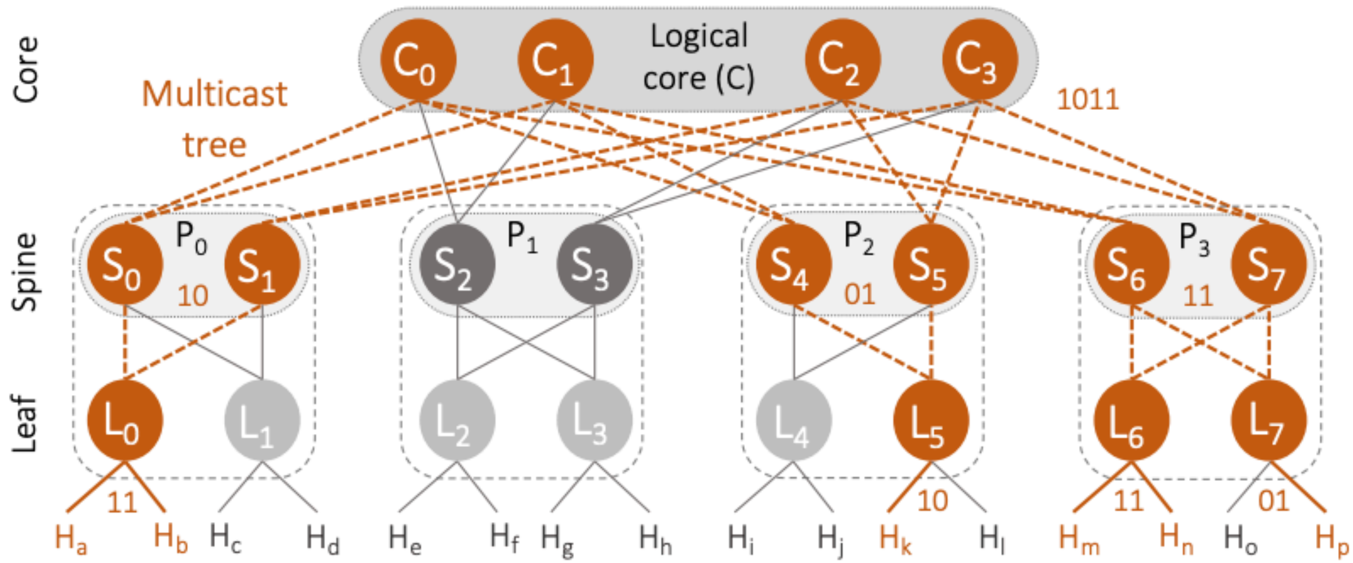
Core

Spine

Leaf

3 Switches within a layer with **same** ports **share a bitmap**

Encoding a Multicast Policy in Elmo



Encoding a Multicast Policy in Elmo

A multicast group encoded as
a list of **(Switch, Ports)** pairs

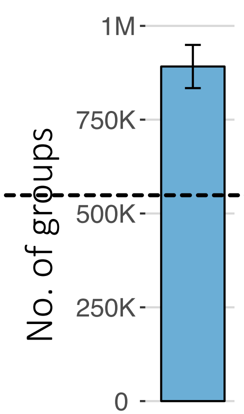
```
Switch 1: [Bitmap]
Switch 2,3: [.. .. ..]
Switch 4: [.. .. .x ..]
Switch 5: [.x .. .. ..]
```

Core

Leaf

3 Switches within a layer with **same** ports **share a bitmap**

Modern **commodity switches** can **parse** packet headers of **512 bytes**



For a data center with:
- **628** switches
- **325** bytes header space

Supports **890,000** groups!

Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

4 Switches within a layer with N **different** ports **share a bitmap**

```
Switch 1: [Bitmap]
Switch 2,3: [.. .. ..]
Switch 4: [.. .. .x ..]
Switch 5: [.x .. .. ..]
```

Core

Spine

Leaf

Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

```
Switch 1: [Bitmap]
Switch 2,3: [.. .. ..]
Switch 4,5:
        [.x .. .x ..]
```

Core

Spine

Leaf

4 Switches within a layer with **N different** ports **share a bitmap**

Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

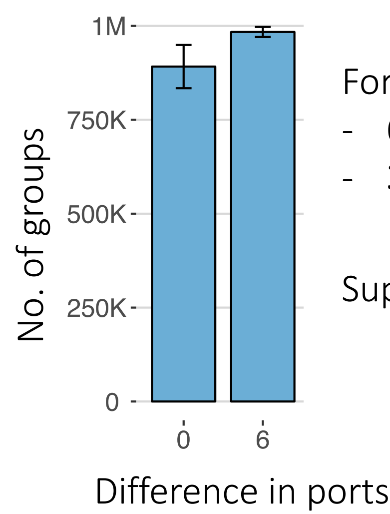
```
Switch 1: [Bitmap]
Switch 2,3: [.. .. ..]
Switch 4,5:
        [.x .. .x ..]
```

Core

Spine

Leaf

4 Switches within a layer with **N different** ports **share a bitmap**



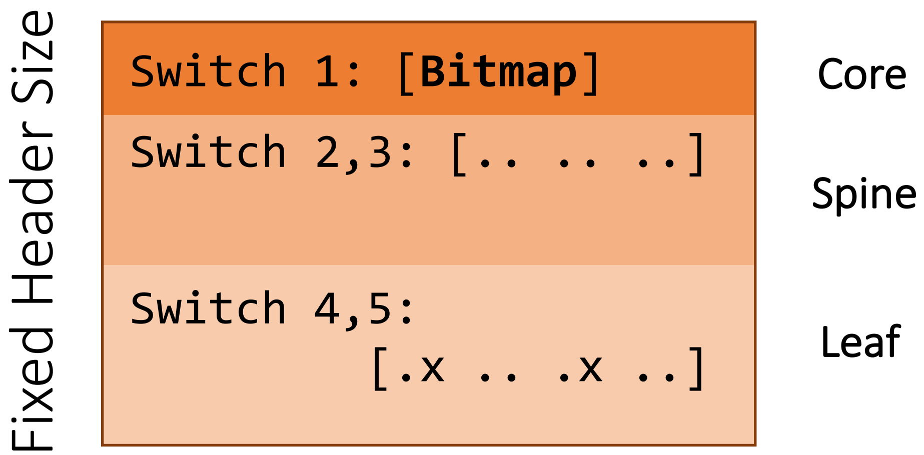
For a data center with:

- **628** switches
- **325** bytes header space

Supports **980,000** groups!

Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

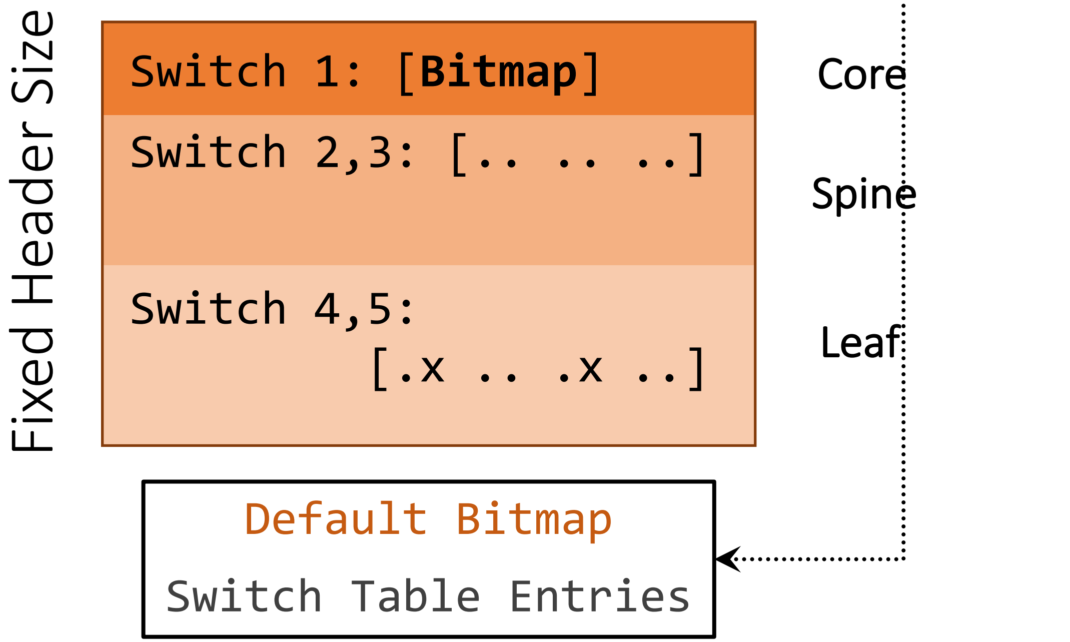


4 Switches within a layer with **N different** ports **share a bitmap**

Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

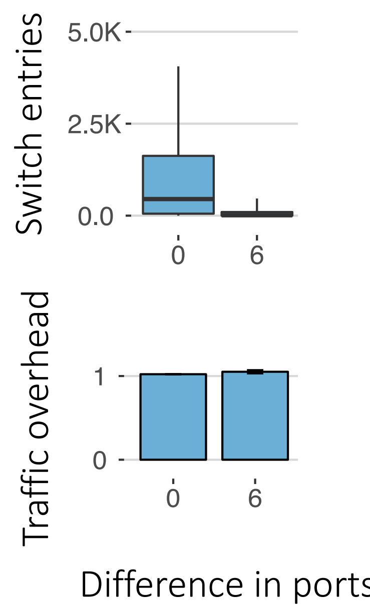
5 Use **switch entries** and a **default bitmap** for **larger groups**



Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

5 Use **switch entries** and a **default bitmap** for **larger groups**



For a data center with:

- **628** switches
- **325** bytes header space

Encoding a Multicast Policy in Elmo

A multicast group encoded as a list of **(Switch, Ports)** pairs

Fixed Header Size

```
Switch 1: [Bitmap]
Switch 2,3: [.. .. ..]
Switch 4,5:
          [.x .. .x ..]
```

Core

Spine

Leaf

```
Default Bitmap
Switch Table Entries
```

1. Encode switch ports as a **bitmap**
2. Group switches into **layers**

Switches within a layer with:

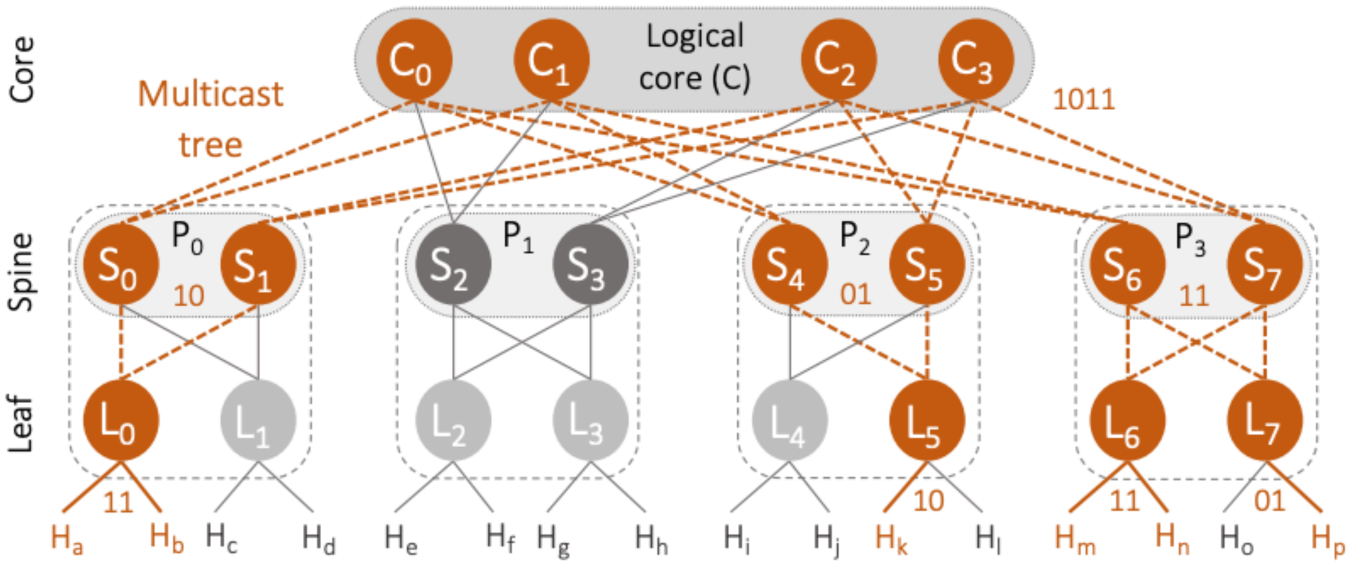
3. - **same** ports **share a bitmap**
4. - **N different** ports **share a bitmap**
5. Use **switch entries** and a **default bitmap** for **larger groups**

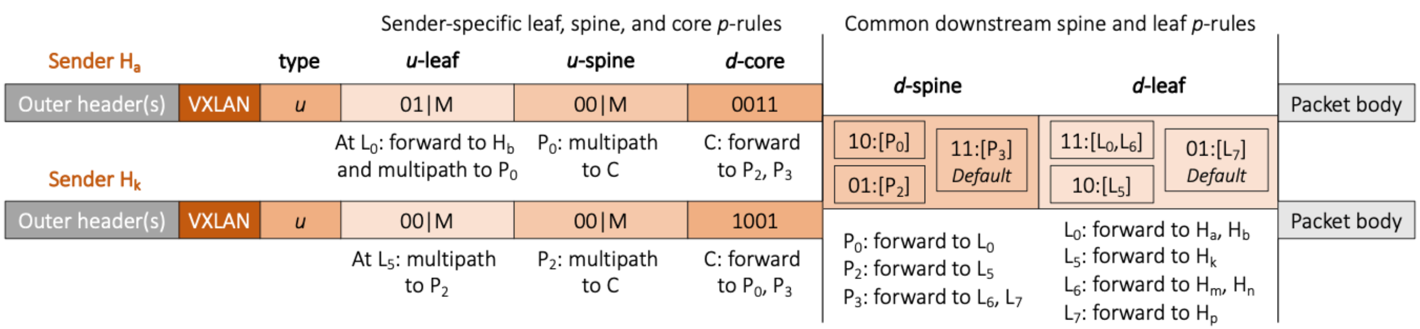
For a data center with:

- **628** switches
- **325** bytes header space

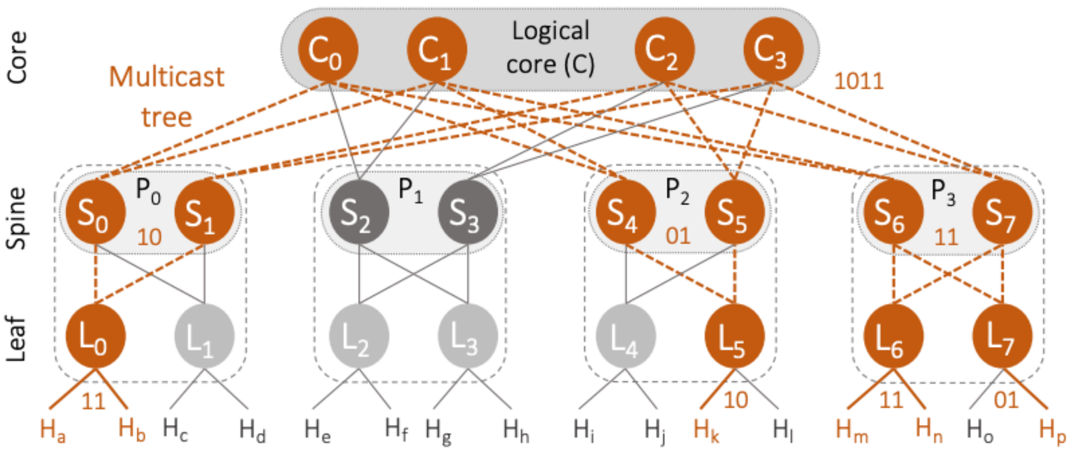
Supports **a Million** groups!

Encoding a Multicast Policy in Elmo





Encoding a Multicast Policy in Elmo

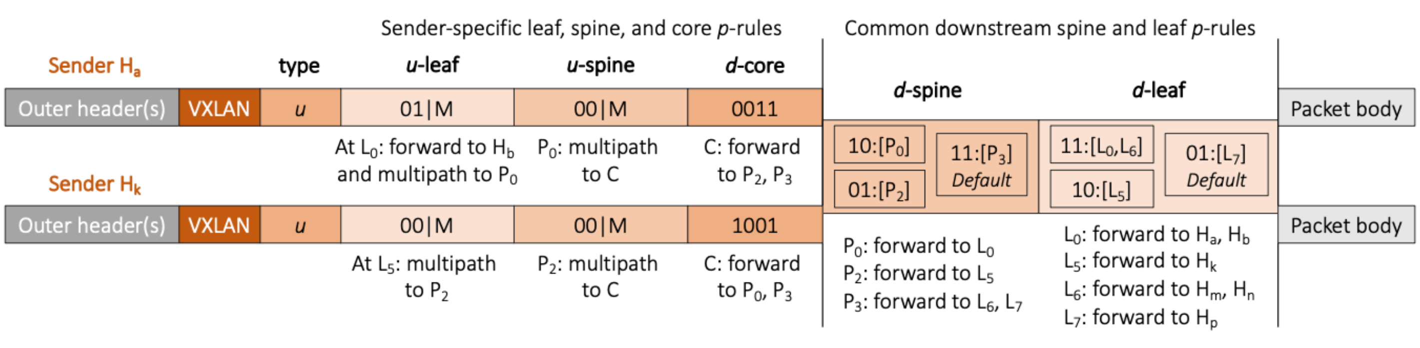


Downstream spine and leaf assignments with varying degree of redundancy (R) and *s*-rules.

R = 0		R = 2
#s-rules = 0	*#s*-rules = 1	*#s*-rules = 0, 1
(*p*) 10:[P_0] (*p*) 01:[P_2] (*d*) 11:[P_3]	(*p*) 10:[P_0] (*p*) 01:[P_2] (*s*) 11:[P_3]	(*p*) 10:[P_0] (*p*) 11:[P_2,P_3]
(*p*) 11:[L_0,L_6] (*p*) 10:[L_5] (*d*) 01:[L_7]	(*p*) 11:[L_0,L_6] (*p*) 10:[L_5] (*s*) 01:[L_7]	(*p*) 11:[L_0,L_6] (*p*) 11:[L_5,L_7]

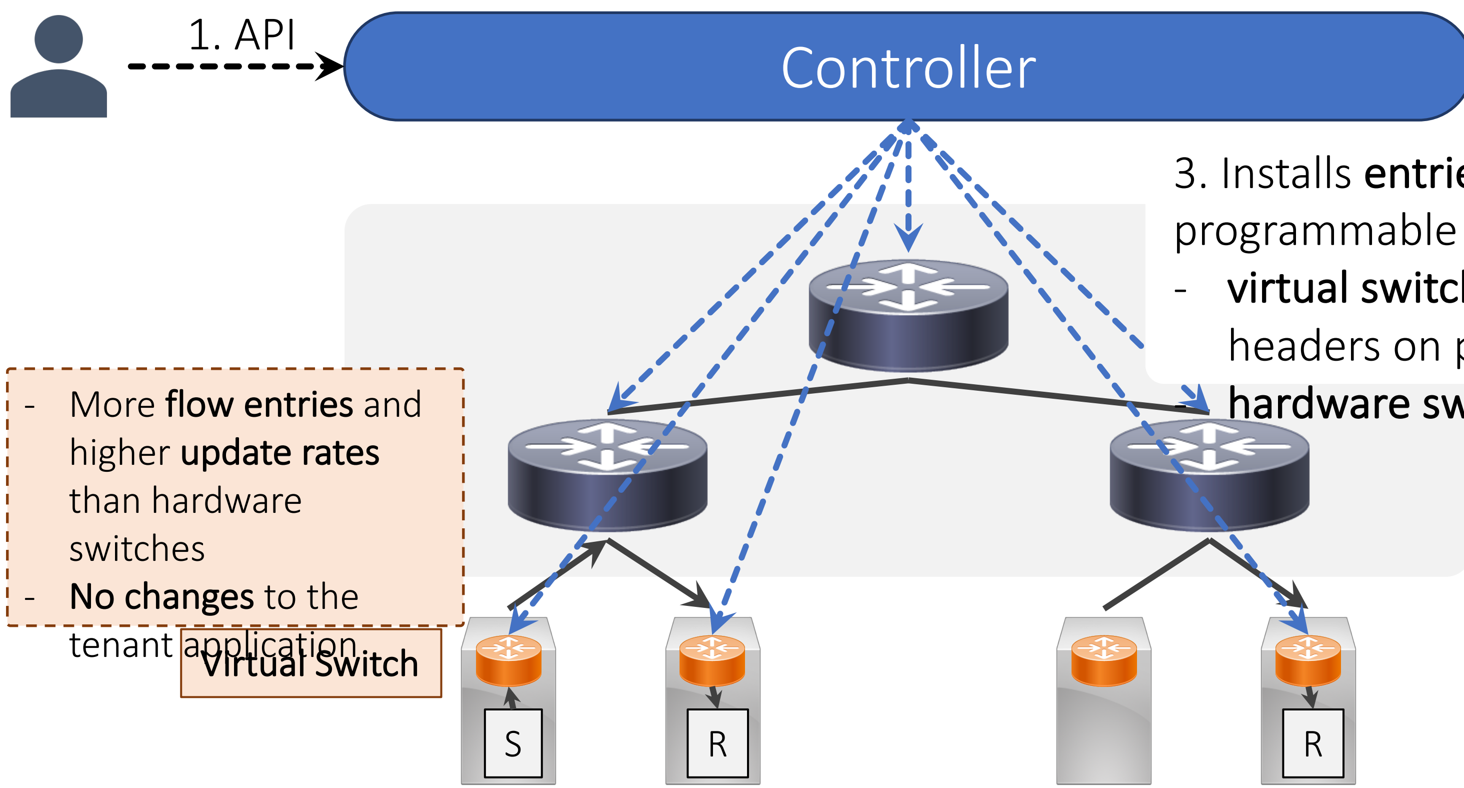
#p-rules = 2 (and max two switches per *p*-rule)

legends: *p*-rule (*p*), *s*-rule (*s*), and default *p*-rule (*d*)



Processing a Multicast Policy in Elmo

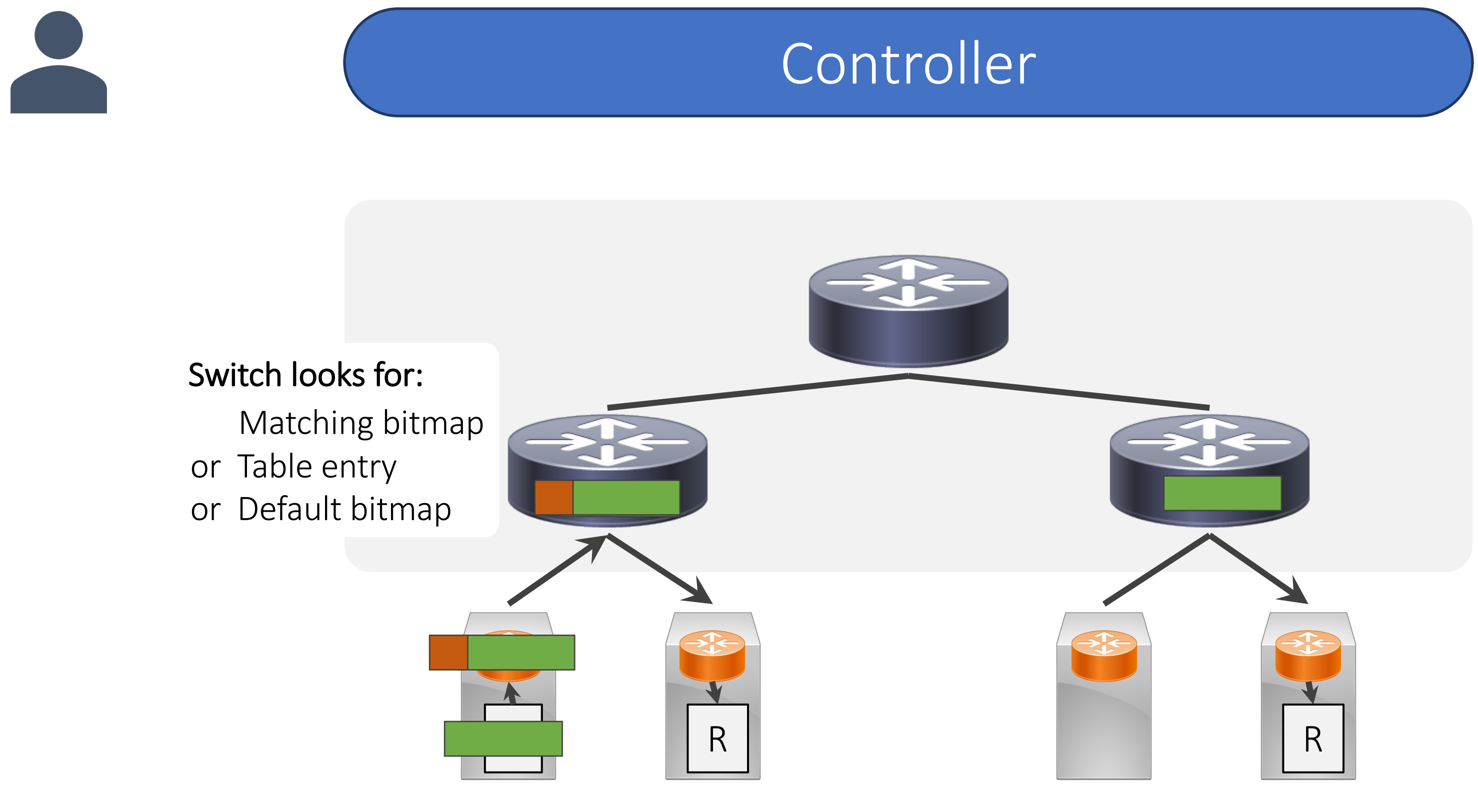
2. Computes the multicast policy



3. Installs **entries** in programmable
- **virtual switches** to push Elmo headers on packets
- **hardware switches**

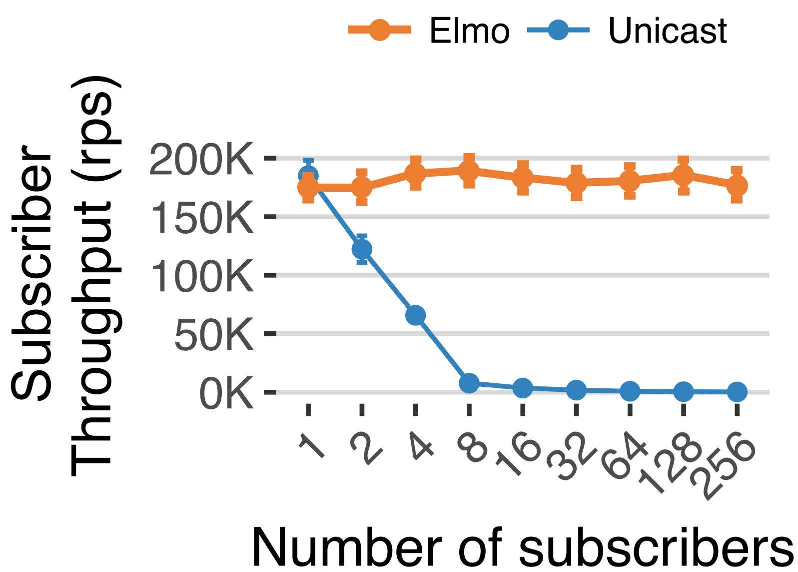
- More **flow entries** and higher **update rates** than hardware switches
- **No changes** to the tenant application

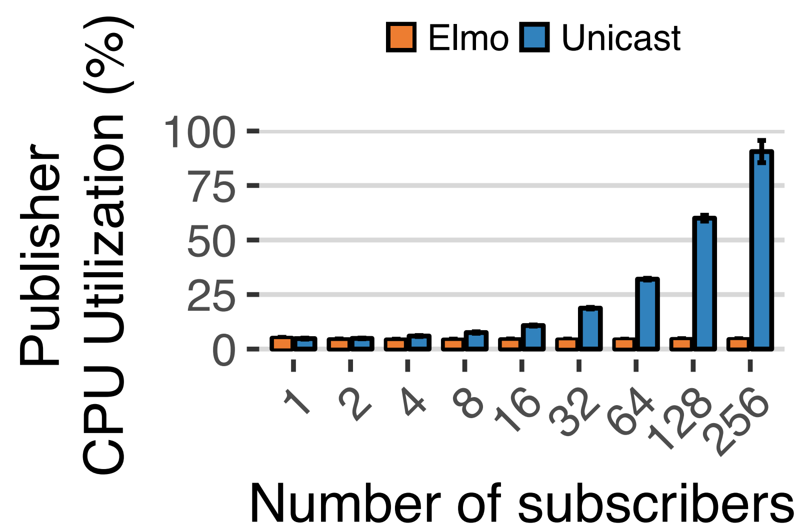
Processing a Multicast Policy in Elmo



Implemented using P4 on a Barefoot Tofino Switch

Applications Run Without Performance Overhead





Conclusion

Elmo

Source Routed Multicast for Public Clouds

- Designed for multi-tenant data centers
- Compactly encodes multicast policy inside packets
- Operates at hardware speed using programmable data planes

Is this a good usecase of programmable dataplanes?

What are the limitations?

Other networking usecases

- Load balancing:
  - HULA: Scalable Load Balancing Using Programmable Data Planes, SOSR'16
- Congestion control:
  - Evaluating the Power of Flexible Packet Processing for Network Resource Allocation, NSDI'17
    - *Support RCP and XCP on programmable switches*
  - HPCC: High Precision Congestion Control, SIGCOMM'19
    - *Obtain precise link information for congestion control*
- A new protocols for more efficient L2 switching
  - The Deforestation of L2, SIGCOMM'16
- And others…