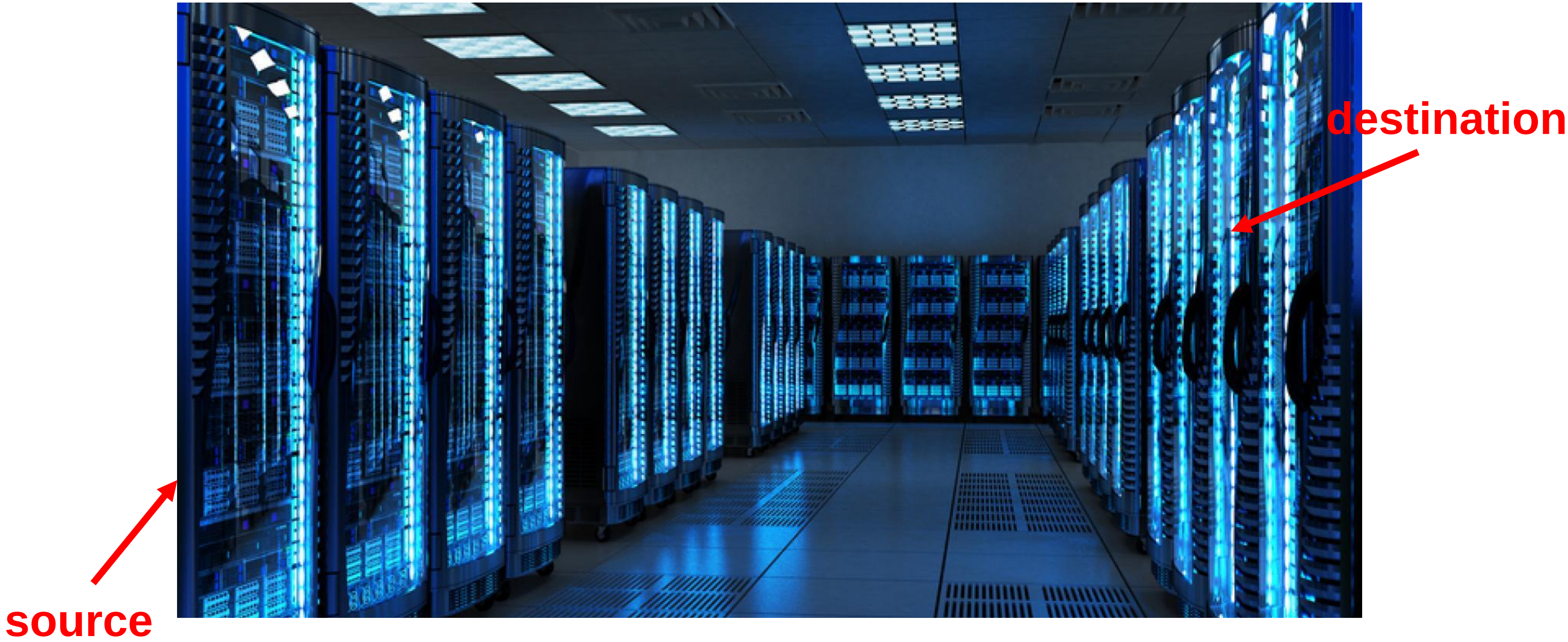
Data Structures
Programming Project #3

Data Center

- A data center consists of multiple severs
- The servers are connected by switches in a local area network



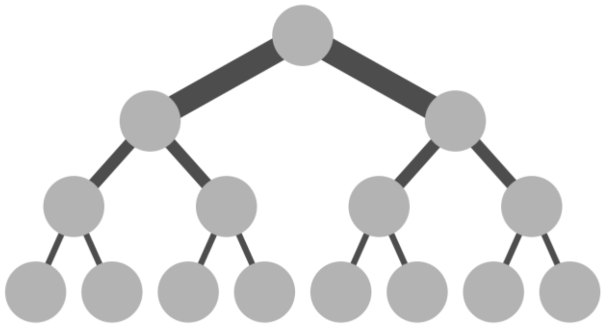
Servers in Data Centers

- Rack servers and blade serve
- Pros and cons

http://techgenix.com/tower-vs-rack-vs-blade-servers/

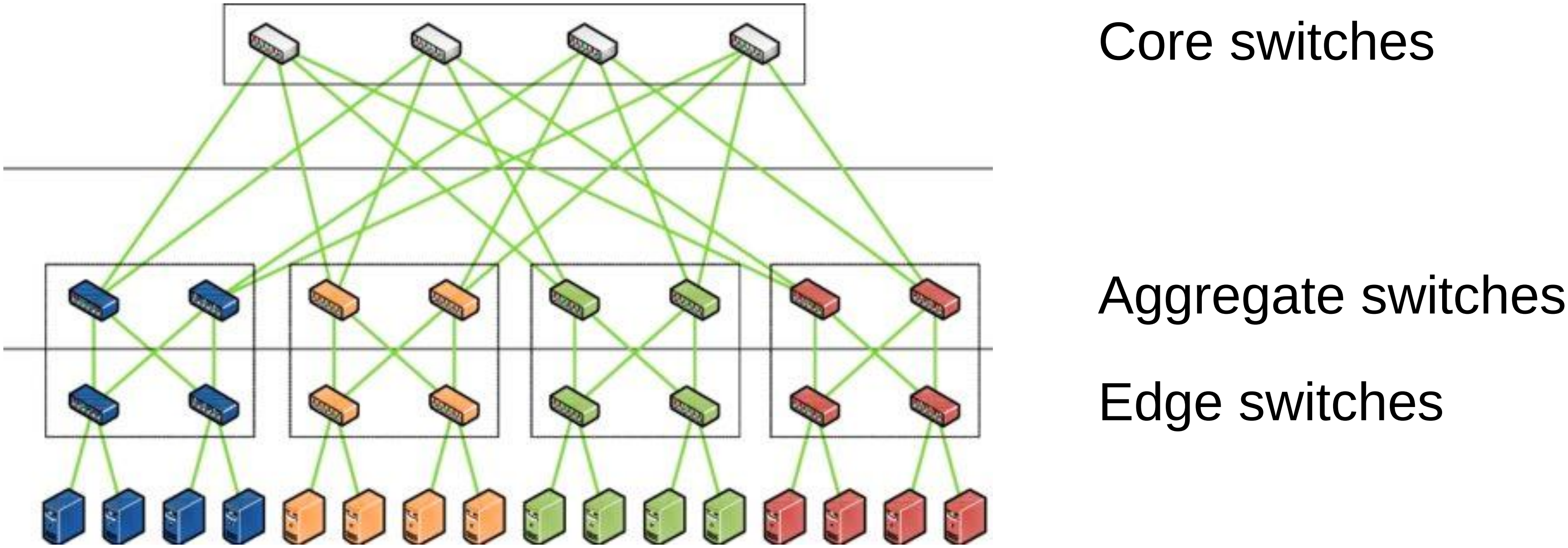
https://www.racksolutions.com/news/data-center-optimization/blade-server-vs-rack-server/

Network Architectures



- The different topologies of networks connecting the servers are designed for different purposes

UCSD Fat-Tree data center architecture

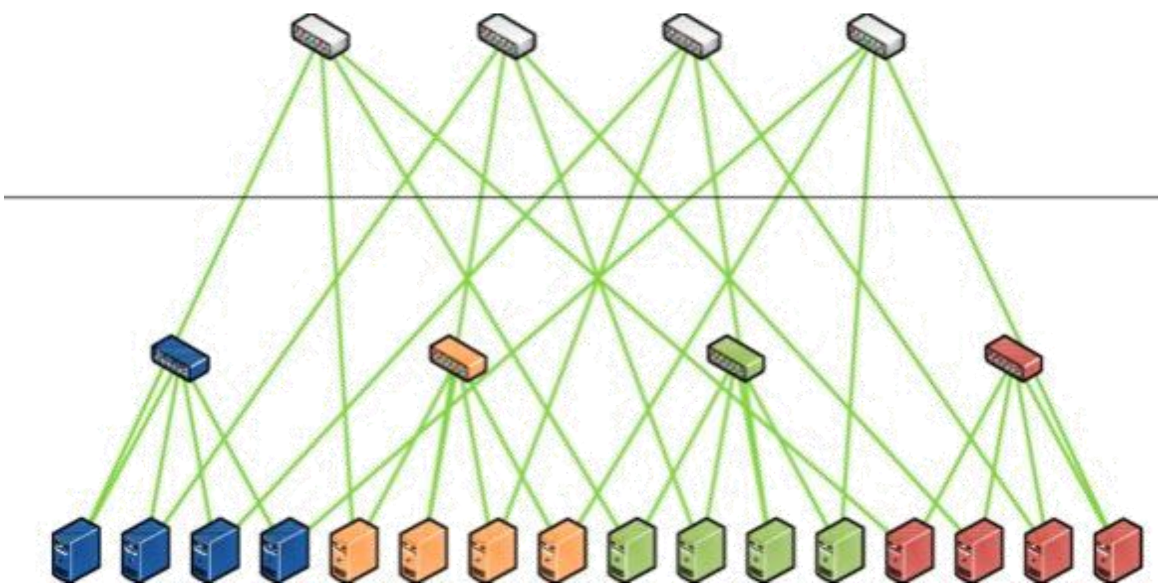


"A Scalable, Commodity Data Center Network Architecture," in ACM SIGCOMM 2008

Network Architectures

- The different topologies of networks connecting the servers are designed for different purposes

Microsoft BCube data center architecture



Level 1 switches

Level 2 switches

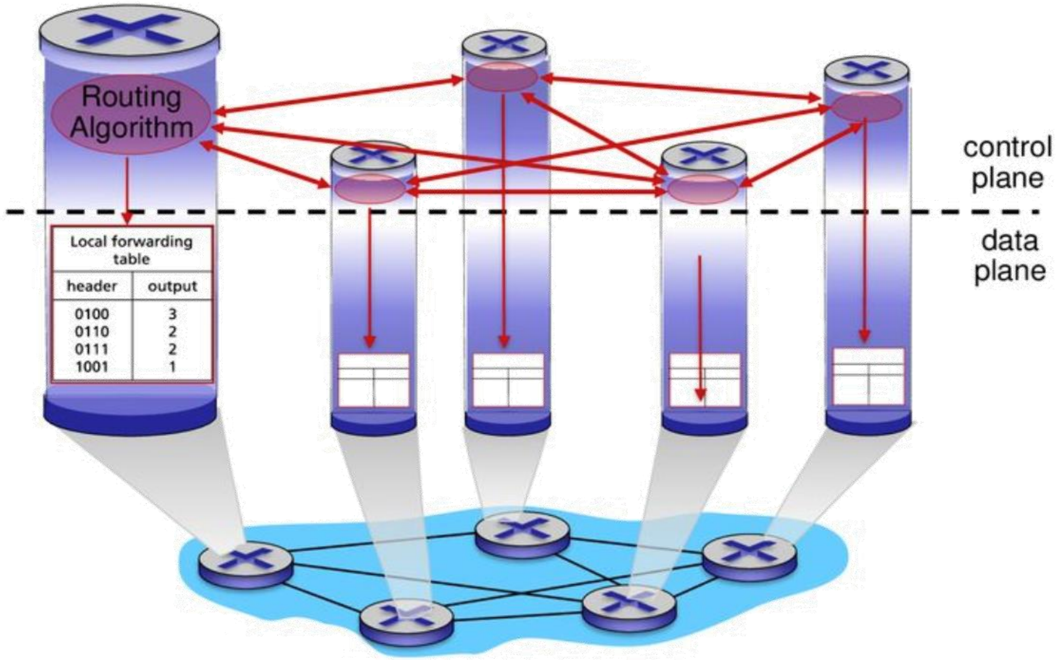
“BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers,” in ACM SIGCOMM 2009

Switches

- Each switch has multiple ports
- Receive and forward the packets from a port to another port

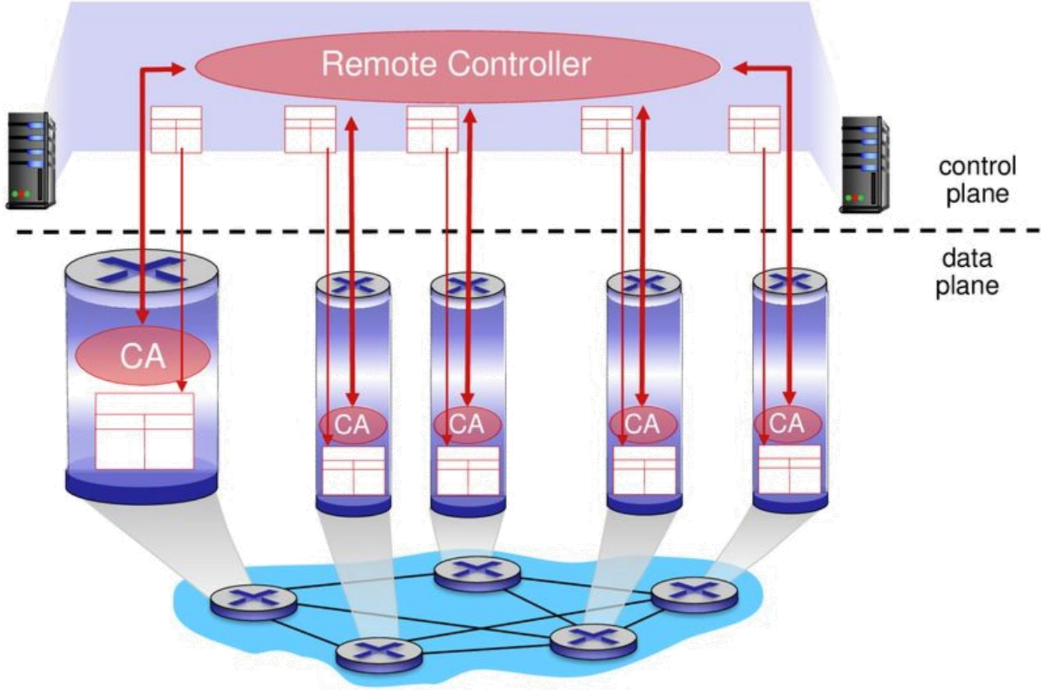
Switches

- Switches run distributed routing algorithms to decide routing paths
- Each switch maintains a routing table
- The routing paths are stored in the table

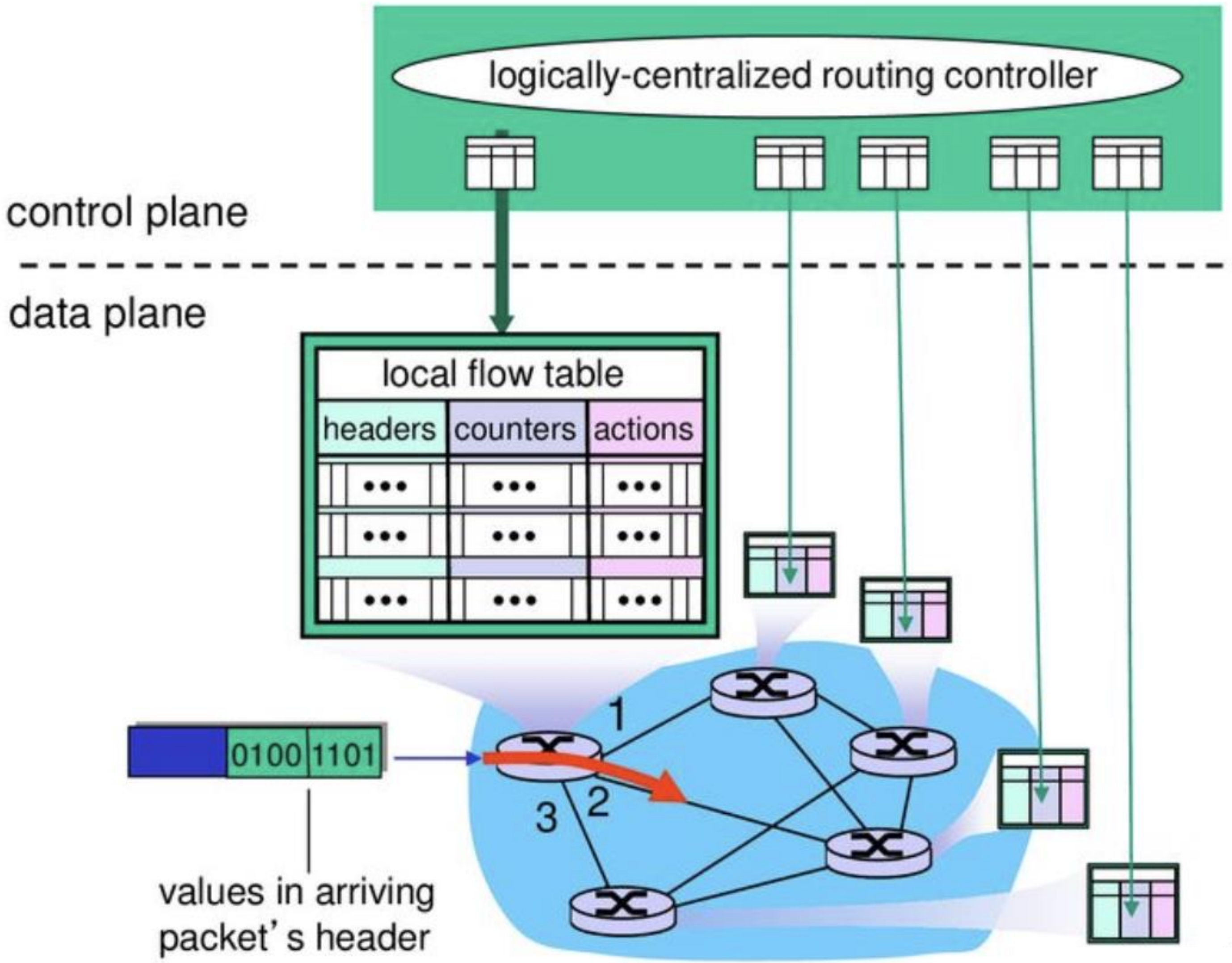


Switches

- Distributed algorithms cannot optimize the routing efficiency globally
- A centralized controller is introduced – software-defined networking (SDN)

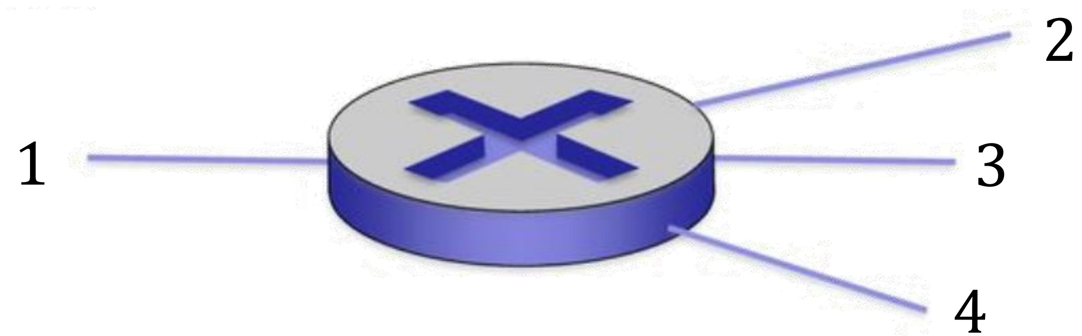


Installing Rules in the Switches



Patterns and Actions in TCAM
Ternary (三態) Content Addressable Memory

- 0 and 1 → 0, 1 and "don't care"
- Pattern: match values in packet header
- Actions for matched packet: drop, forward,…
- Counters: #packets is matched to each pattern



* : wildcard

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

Pros and Cons of TCAM

- Compare a search key in parallel against all entries
- Report the first matching entry
- Allow ‘don’t care’

- Dense circuits → expensive, power-hungry, hot
- The on-chip TCAM sizes are typically limited to a few thousand entries

- Compress or decompose it!

“On the effect of forwarding table size on SDN network utilization,” in IEEE INFOCOM 2014

Entry Utilization Minimization

- The original table → 9 entries
- The modified table by wildcard (*,*) → 7 entries

Flow	Output port
(0, 4)	Port-4
(0, 5)	Port-5
(0, 6)	Port-5
(1, 4)	Port-6
(1, 5)	Port-4
(1, 6)	Port-6
(2, 4)	Port-4
(2, 5)	Port-5
(2, 6)	Port-6

(a) Without Compression

Flow	Output port
(0, 5)	Port-5
(0, 6)	Port-5
(1, 4)	Port-6
(1, 6)	Port-6
(2, 5)	Port-5
(2, 6)	Port-6
$(*, *)$	Port-4

(b) With $(*, *)$ rule

Entry Utilization Minimization

- The original table → 9 entries
- The modified table by wildcard (n,*) → 6 entries

Flow	Output port
(0, 4)	Port-4
(0, 5)	Port-5
(0, 6)	Port-5
(1, 4)	Port-6
(1, 5)	Port-4
(1, 6)	Port-6
(2, 4)	Port-4
(2, 5)	Port-5
(2, 6)	Port-6

(a) Without Compression

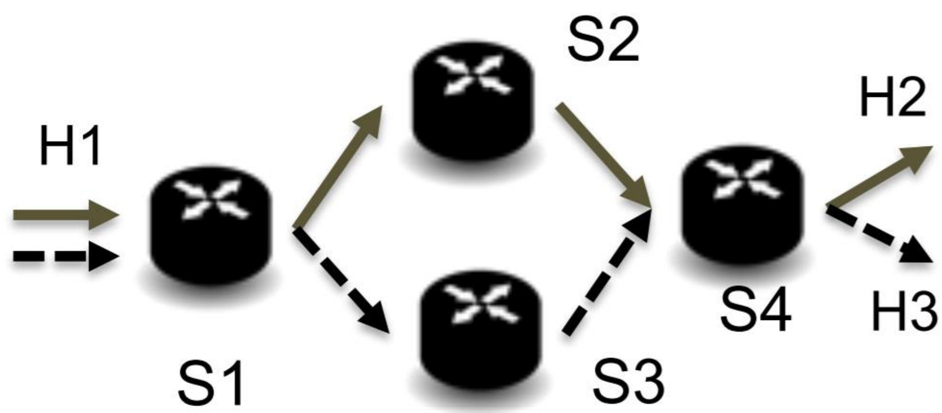
Flow	Output port
(0, 4)	Port-4
(1, 5)	Port-4
(2, 4)	Port-4
(2, 5)	Port-5
$(0, *)$	Port-5
$(*, *)$	Port-6

(c) With $(n, *)$ rule

"One Big Switch" Abstraction in SDN

$$r_1 : (\text{dst\_ip} = 00*, \text{ingress} = H_1 : \text{Permit}, \text{egress} = H_2)$$
$$r_2 : (\text{dst\_ip} = 01*, \text{ingress} = H_1 : \text{Permit}, \text{egress} = H_3)$$

(a) An example endpoint policy E



(b) An example routing policy R

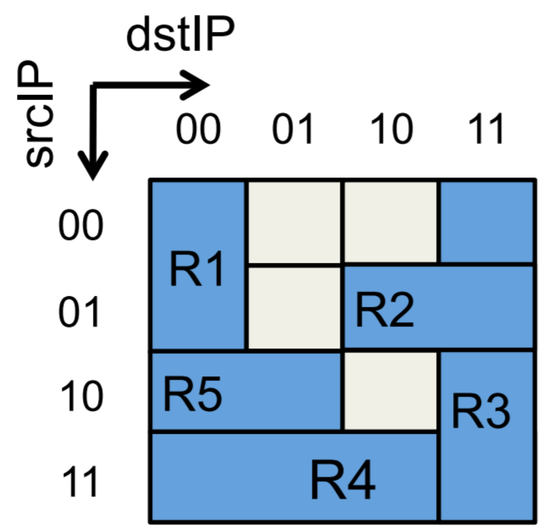
$$P_1 = s_1 s_2 s_4, D_1 = \{\text{dst\_ip} = 00*\}$$
$$P_2 = s_1 s_3 s_4, D_2 = \{\text{dst\_ip} = 01*\}$$

Access control policy in the table
Prefix or Exact Matching

Rule	Source IP	Destination IP	Action
R1	0*	00	Permit
R2	01	1*	Permit
R3	*	11	Drop
R4	11	*	Permit
R5	10	0*	Permit
R6	*	*	Drop

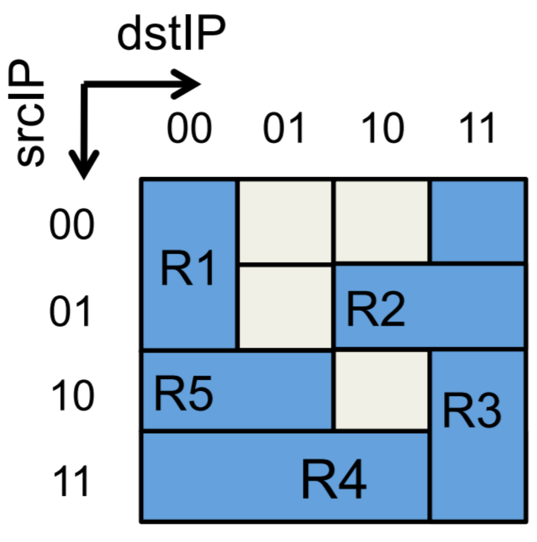
Rectangular representation

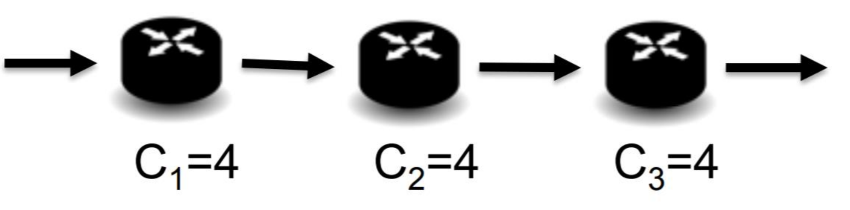
Rule	Source IP	Destination IP	Action
R1	0*	00	Permit
R2	01	1*	Permit
R3	*	11	Drop
R4	11	*	Permit
R5	10	0*	Permit
R6	*	*	Drop



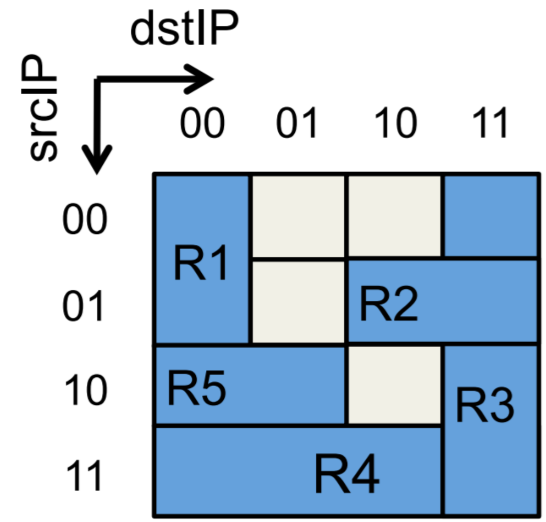
Implementation of "One Big Switch" Abstraction

- Limited capacity of a single switch
  e.g., capacity = 4 but # rules = 6
- Multiple switches are regarded as a big switch
  e.g., total capacity = 4 x 3 = 12

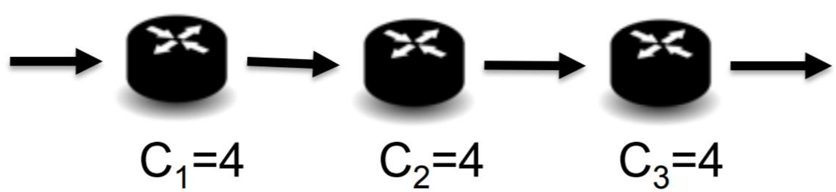


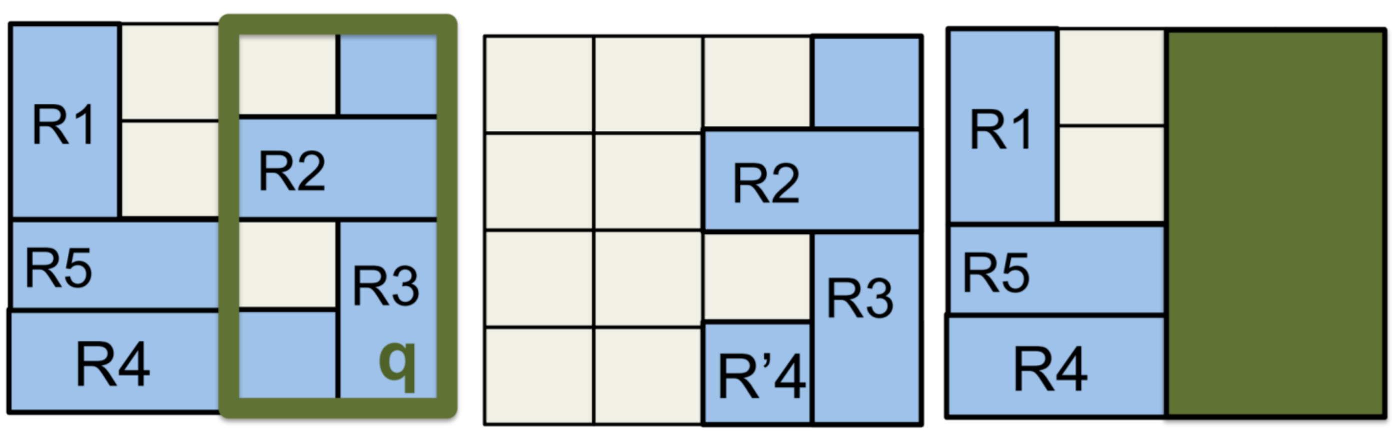


Placing Rules Along a Path



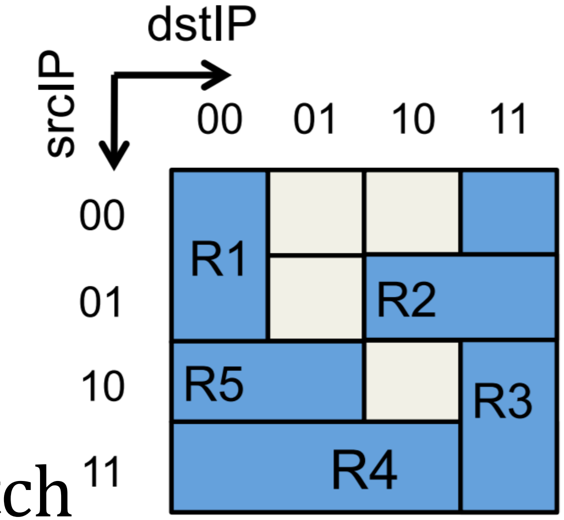
- Cover Phase:
- Find a rectangle and identify the internal rules (e.g., R2, R3) and overlapping rules (R4, R6)



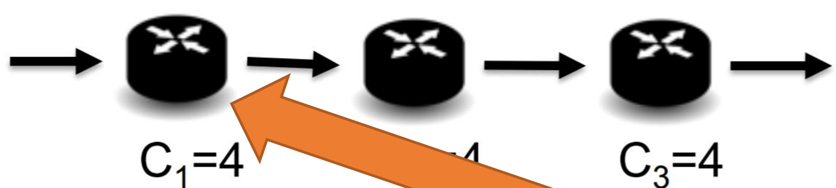


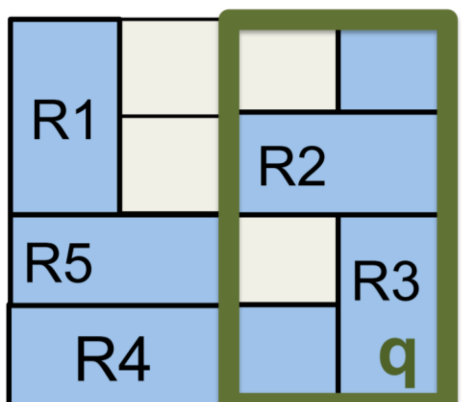
(a) Cover q (b) Switch s_1 (c) After s_1

Placing Rules Along a Path

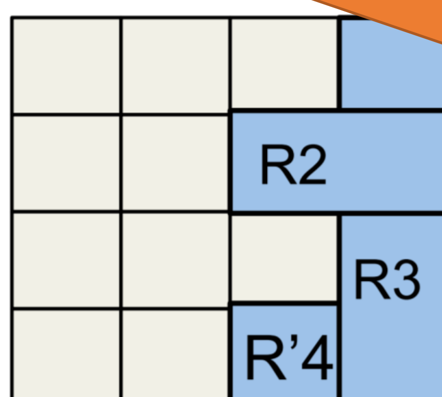


- Pack Phase:
- Place the rules on the current switch (e.g., the first switch)





(a) Cover q



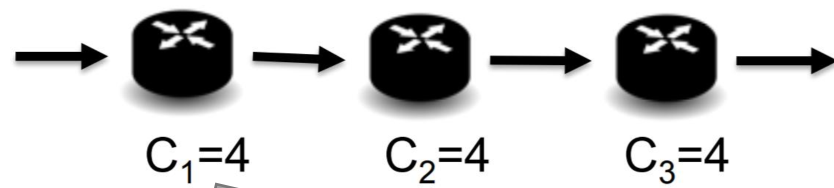
(b) Switch s_1

$$r_1 : (q \wedge R2.p, R2.a)$$
$$r_2 : (q \wedge R3.p, R3.a)$$
$$r_3 : (q \wedge R4.p, R4.a)$$
$$r_4 : (q \wedge R6.p, R6.a)$$

(a) E_q

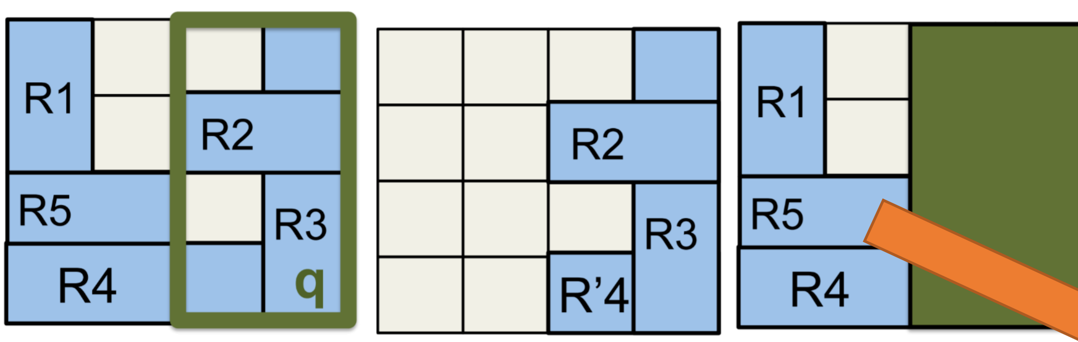
Placing Rules Along a Path

- Replace Phase
- Rewrite the rules to avoid re-processing the packets in the rectangle



$r_1 : (q \wedge R2.p, R2.a)$
$r_2 : (q \wedge R3.p, R3.a)$
$r_3 : (q \wedge R4.p, R4.a)$
$r_4 : (q \wedge R6.p, R6.a)$

(a) E_q



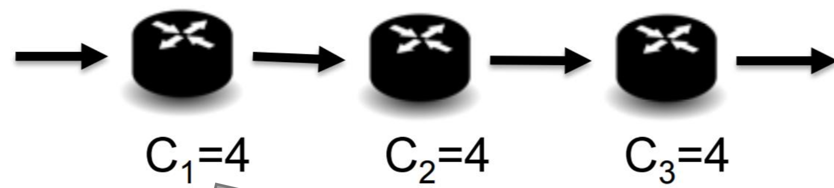
(a) Cover q (b) Switch s_1 (c) After s_1

$\boxed{r_1 : (q, \mathrm{Fwd})}$
$r_2 : (R1.p, R1.a)$
$r_3 : (R4.p, R4.a)$
$r_4 : (R5.p, R5.a)$
$r_5 : (R6.p, R6.a)$

(b) New rule list

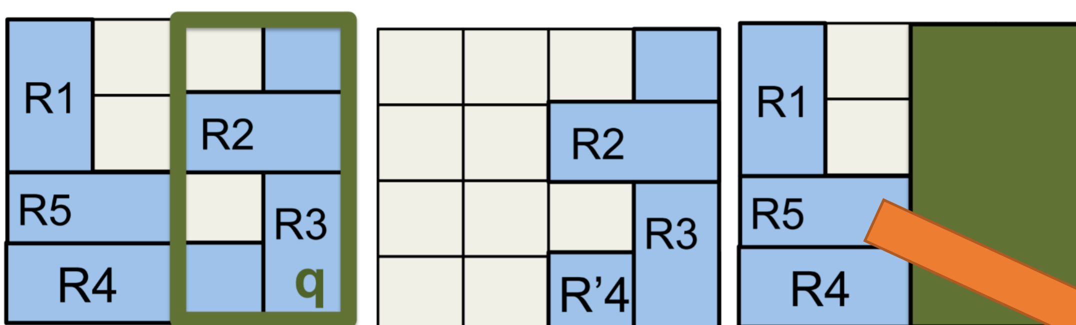
Drawback

- Additional rules to avoid re-processing the packets in the rectangle
- Could be multiple rectangles



$r_1 : (q \wedge R2.p, R2.a)$
$r_2 : (q \wedge R3.p, R3.a)$
$r_3 : (q \wedge R4.p, R4.a)$
$r_4 : (q \wedge R6.p, R6.a)$

(a) E_q



(a) Cover q (b) Switch s_1 (c) After s_1

$\boxed{r_1 : (q, \mathrm{Fwd})}$
$r_2 : (R1.p, R1.a)$
$r_3 : (R4.p, R4.a)$
$r_4 : (R5.p, R5.a)$
$r_5 : (R6.p, R6.a)$

(b) New rule list

Programming Project #3: Placing Rules Along a Path

- Input:
  - Numbers of switches
  - IDs and capacities of switches
  - Source and destination
  - Rules
- Procedure:
  - Cover, pack, and replace phase
- Output:
  - The shortest routing path
  - The rules in each switches
- The grade is inversely proportional to the number of switches with rules (except routing)

Programming Project #3: Placing Rules Along a Path

- Input:
  - Numbers of switches
  - IDs and capacities of switches
  - Source and destination
  - Rules
- Procedure:
  - Cover, pack, and replace phase
- Output:
  - The shortest routing path
  - The rules in each switches
- The grade is inversely proportional to the number of switches with rules (except routing)

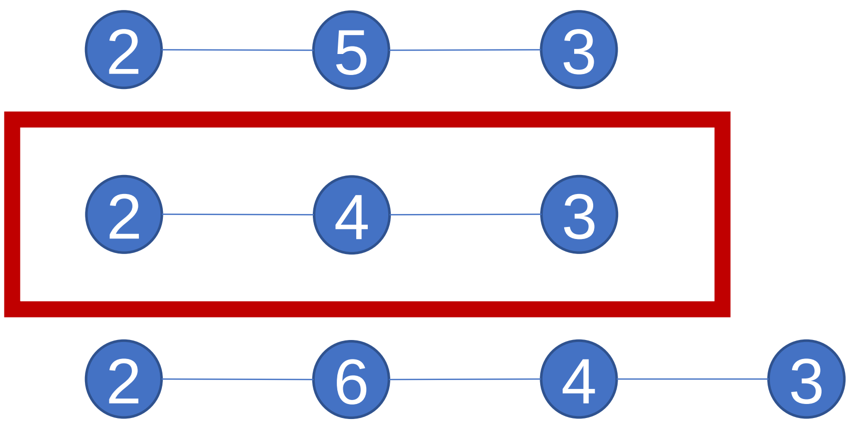
要棄選了嗎?

Programming Project #3: Cost-Efficient Covering & Packing

- Input:
  - Numbers of switches
  - IDs and capacities of switches
  - Source and destination
- Procedure:
  - Cover, pack, and replace phase
- Output:
  - The shortest routing path
  - The rules in each switches
- The algorithm is given
  You have to implement the algorithm

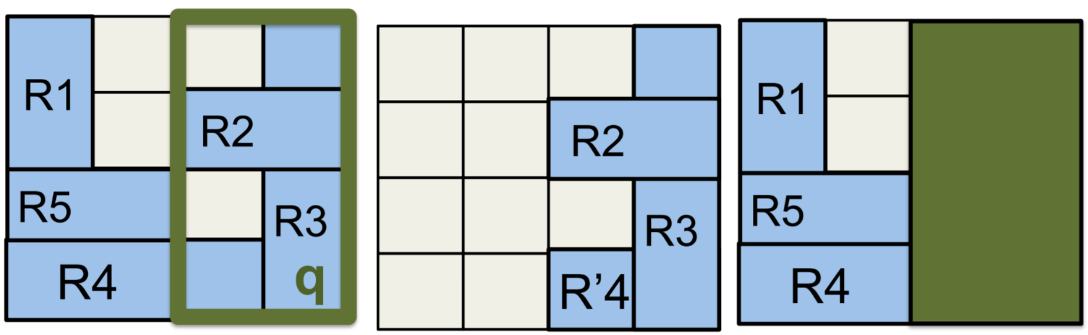
Find the Path

- Given:
  A pair of an ingress switch and an egress switch
- Find the shortest path by Dijkstra Algorithm
- For tie breaking, we give the priority to the one:
  1. with the smaller hop count
  2. with the smaller switch ID in the sequence



The Algorithm for Placing Rules Along a Path

- Cover Phase
- If the remaining size is sufficient → place all the rules
- Otherwise, pick the rectangle with the maximum utility
- **utility(q) $= \frac{\#\text{internal rules} - 1}{\#\text{overlapping rules} + 1}$**

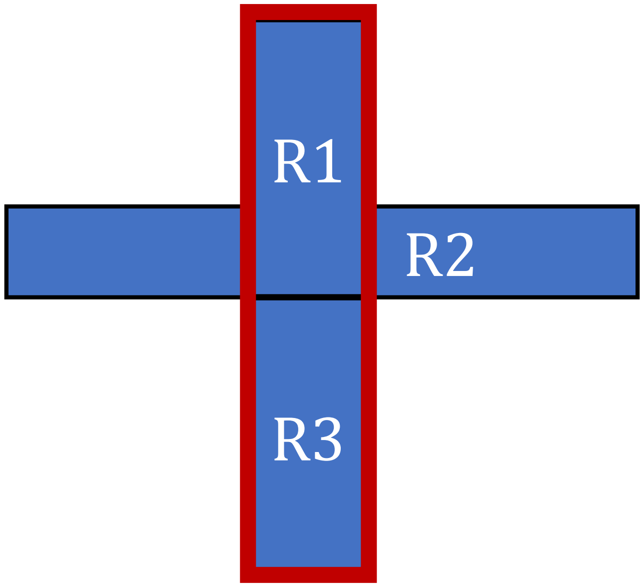


(a) Cover q (b) Switch s_1 (c) After s_1

- E.g., internal rules (e.g., R2, R3 are fully covered by q) and overlapping rules (R4, R6 are partially covered by q)
- utility(q) $= \frac{2-1}{2+1} = \frac{1}{3}$

Remark for Overlapping Rules

- Assume that the default rule is R4 (*, *, Drop)
- R2 and R4 are not considered as overlapping rules in the following red rectangle



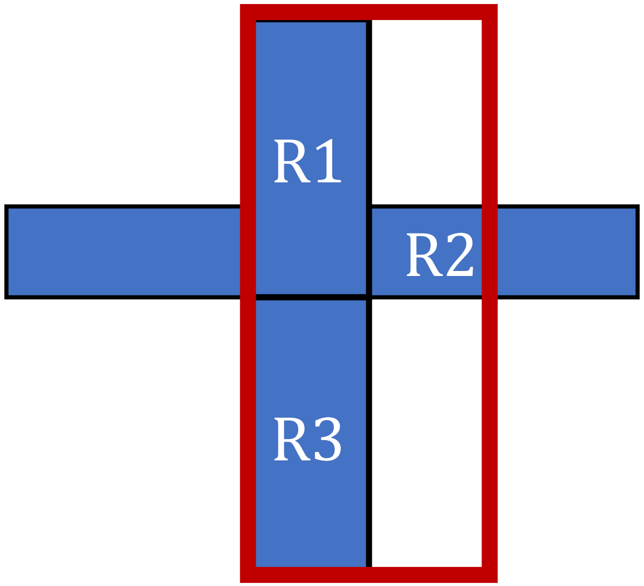
$\boldsymbol{r_2}$ and $\boldsymbol{r_4}$ are removed because they are fully covered by $\boldsymbol{R1}$ and $\boldsymbol{R3}$ in $\boldsymbol{q}$

$$\frac{2-1}{1} = \frac{1}{1}$$

$$\boldsymbol{r_1: (q \wedge R1.p, R1.a)}$$
$$\cancel{\boldsymbol{r_2: (q \wedge R2.p, R2.a)}}$$
$$\boldsymbol{r_3: (q \wedge R3.p, R3.a)}$$
$$\cancel{\boldsymbol{r_4: (q \wedge R4.p, R4.a)}}$$

Remark for Overlapping Rules

- Assume that the default rule is R4 (*, *, Drop)
- In contrast, R2 and R4 are considered as overlapping rules in the following red rectangle



$\boldsymbol{r_2}$ and $\boldsymbol{r_4}$ are considered because they are not fully covered in $\boldsymbol{q}$

$$\frac{2-1}{2+1} = \frac{1}{3}$$

$\boldsymbol{r_1}: (\boldsymbol{q} \wedge \boldsymbol{R1.p}, \boldsymbol{R1.a})$

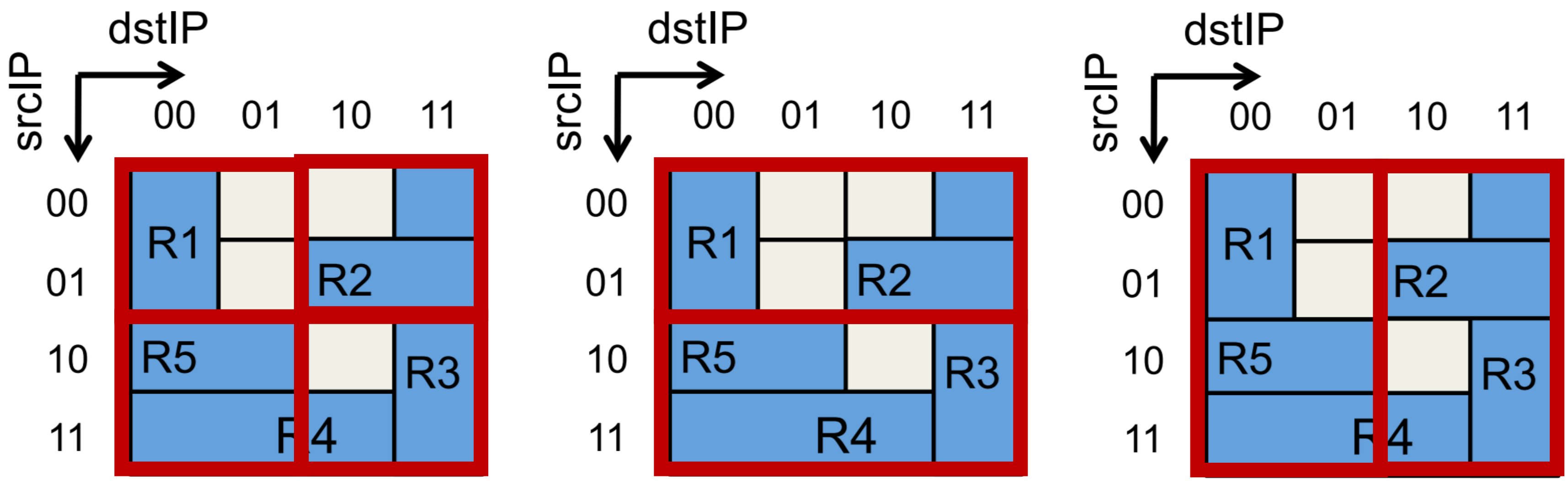
$\boldsymbol{r_2}: (\boldsymbol{q} \wedge \boldsymbol{R2.p}, \boldsymbol{R2.a})$

$\boldsymbol{r_3}: (\boldsymbol{q} \wedge \boldsymbol{R3.p}, \boldsymbol{R3.a})$

$\boldsymbol{r_4}: (\boldsymbol{q} \wedge \boldsymbol{R4.p}, \boldsymbol{R4.a})$

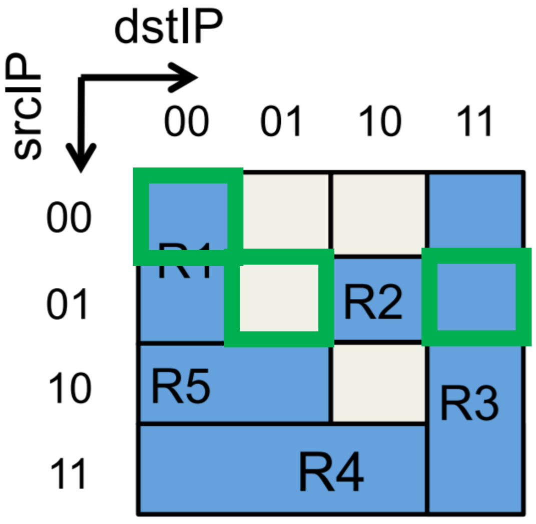
Choose the Rectangle

- You need to examine all the possible rectangles
- #rows = 1, 2, 4, 8, ...
- #columns = 1, 2, 4, 8, ...



Choose the Rectangle

- You need to examine all the possible rectangles
- We only consider the rectangle with utility > 0



$$4 \cdot 4 = 16$$

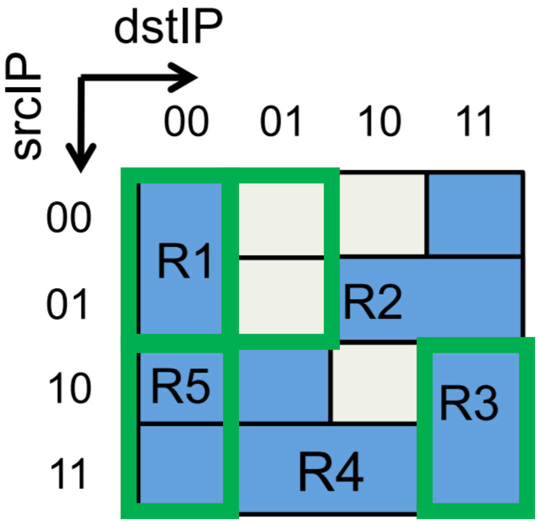
$$\frac{0-1}{1+1} < 0$$

$$\frac{0-1}{1+1} < 0 \qquad \frac{0-1}{1+1} < 0$$

We don't use them

Choose the Rectangle

- You need to examine all the possible rectangles
- We only consider the rectangle with utility > 0



$$2 \cdot 4 = 8$$

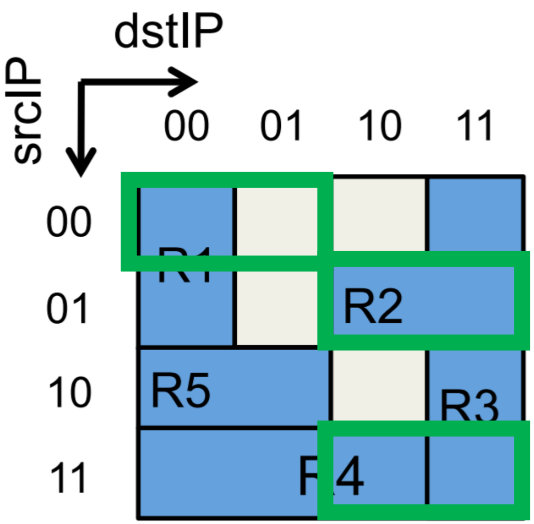
$$\frac{1-1}{1} = 0 \qquad \frac{0-1}{1+1} < 0$$

$$\frac{0-1}{2+1} < 0 \qquad \frac{0-1}{1+1} < 0$$

We don't use them

Choose the Rectangle

- You need to examine all the possible rectangles
- We only consider the rectangle with utility > 0



$$4 \cdot 2 = 8$$

$$\frac{0-1}{2+1} < 0$$

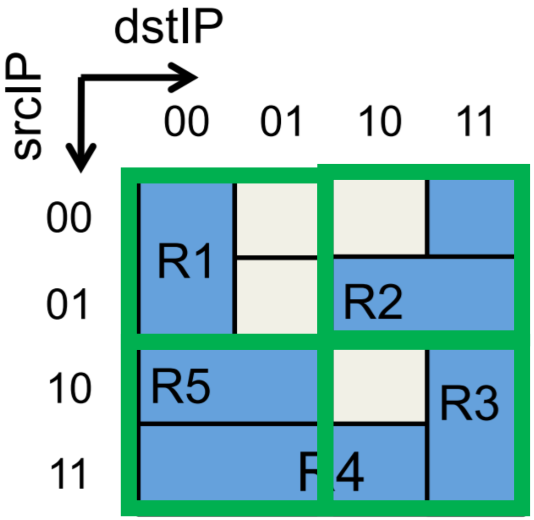
$$\frac{1-1}{1} = 0$$

$$\frac{0-1}{2+1} < 0$$

We don't use them

Choose the Rectangle

- You need to examine all the possible rectangles
- We only consider the rectangle with utility > 0



$$2 \cdot 2 = 4$$

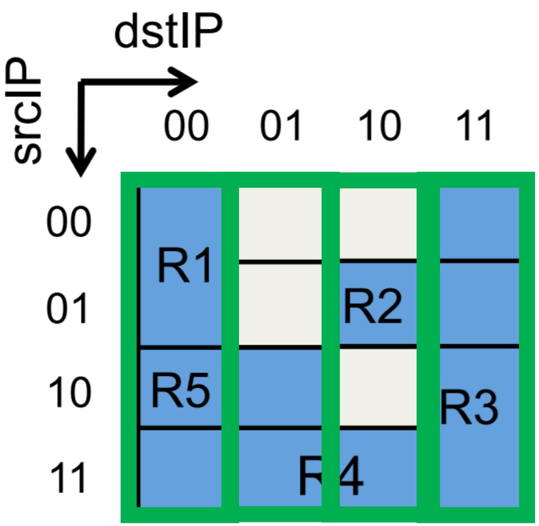
$$\frac{1-1}{1+1} = 0 \qquad \frac{1-1}{2+1} = 0$$

$$\frac{1-1}{1+1} = 0 \qquad \frac{0-1}{3+1} < 0$$

We don't use them

Choose the Rectangle

- You need to examine all the possible rectangles
- We only consider the rectangle with utility > 0



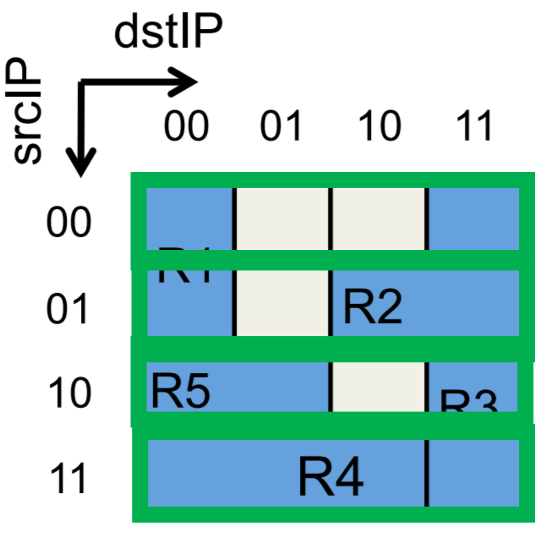
$$\frac{1-1}{2+1}=0 \quad \frac{0-1}{3+1}<0 \quad \frac{0-1}{3+1}<0 \quad \frac{1-1}{1+1}=0$$

$$1 \cdot 4 = 4$$

We don't use them

Choose the Rectangle

- You need to examine all the possible rectangles
- We only consider the rectangle with utility > 0



$$4 \cdot 1 = 4$$

$$\frac{0-1}{3+1} < 0$$

$$\frac{1-1}{2+1} = 0$$

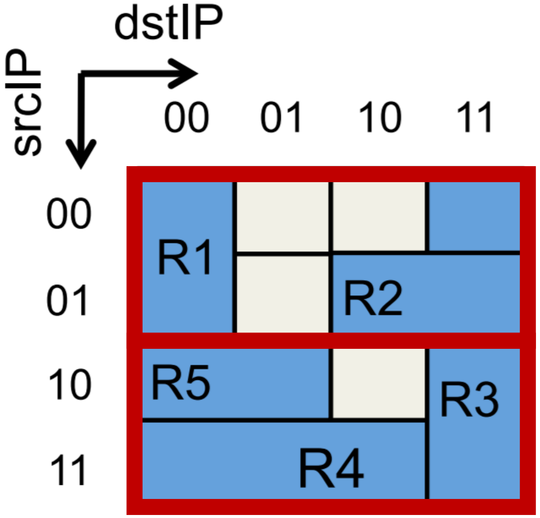
$$\frac{1-1}{2+1} = 0$$

$$\frac{1-1}{1+1} = 0$$

We don't use them

Choose the Rectangle

- You need to examine all the possible rectangles
- We only consider the rectangle with utility > 0



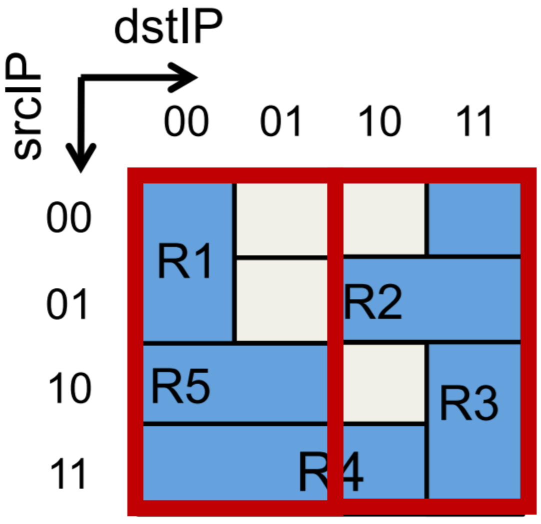
$$2 \cdot 1 = 2$$

$$\frac{2-1}{2+1} = \frac{1}{3}$$

$$\frac{2-1}{2+1} = \frac{1}{3}$$

Choose the Rectangle

- You need to examine all the possible rectangles
- We only consider the rectangle with utility > 0



$$1 \cdot 2 = 2$$

$$\frac{2-1}{2+1} = \frac{1}{3} \qquad \frac{2-1}{2+1} = \frac{1}{3}$$

Number of Examined Rectangles

- Total number of possible rectangles

$$\le 2 \cdot m\left(m + \frac{m}{2} + \frac{m}{4} + \frac{m}{8} \ldots + 1\right) + 2 \cdot \frac{m}{2}\left(m + \frac{m}{2} + \frac{m}{4} + \frac{m}{8} \ldots + 1\right) + \cdots + 2 \cdot \frac{m}{m} \cdot (m + \frac{m}{2} + \frac{m}{4} + \frac{m}{8} \ldots + 1)$$

$$= 2 \cdot m^2\left(1 + \frac{1}{2} + \cdots + \frac{1}{m}\right) + 2 \cdot \frac{m^2}{2}\left(1 + \frac{1}{2} + \cdots + \frac{1}{m}\right) + \cdots + 2 \cdot \frac{m^2}{m} \cdot (1 + \frac{1}{2} + \cdots + \frac{1}{m})$$

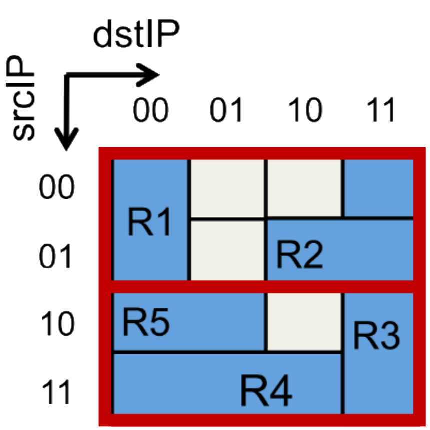
$$\le 2m^2\left(\frac{1}{1-\frac{1}{2}}\right) + 2 \cdot \frac{m^2}{2}\left(\frac{1}{1-\frac{1}{2}}\right) + \cdots + 2 \cdot \frac{m^2}{m}(\frac{1}{1-\frac{1}{2}})$$

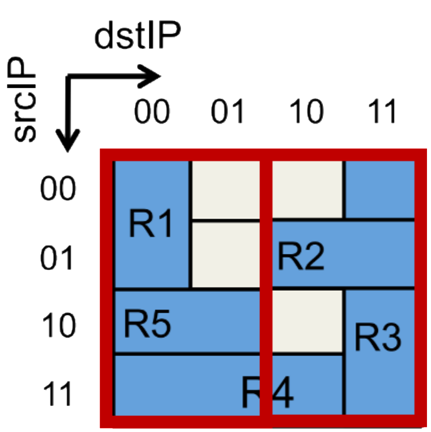
$$= 2m^2 \cdot 2 \cdot (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{m})$$

$$\le 2m^2 \cdot 2 \cdot \left(\frac{1}{1-\frac{1}{2}}\right) = 8m^2 = O(m^2)$$

Choose the Rectangle

- Find the rectangle with the maximum utility
- For tie breaking, we give the priority to the one
    1. with the more internal rule IDs, then…
    2. with the smaller area size, and then
    3. with the smaller internal rule ID in the sequence





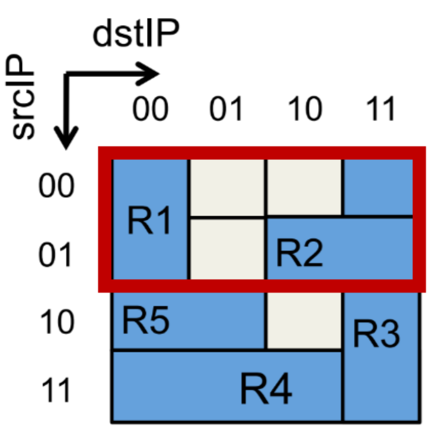
$\boldsymbol{R1}$

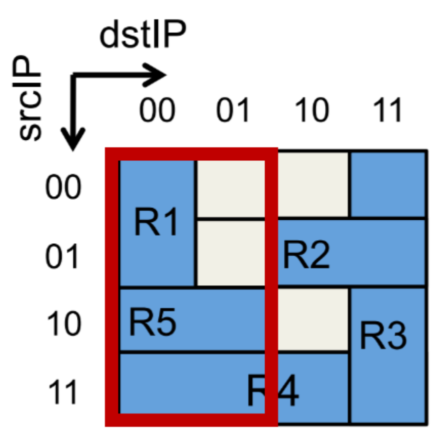
$\boldsymbol{R1}$ $\boldsymbol{R2}$

$\boldsymbol{R4}$

Choose the Rectangle

- Find the rectangle with the maximum utility
- For tie breaking, we give the priority to the one
    1. with the more internal rule IDs, then…
    2. with the smaller area size, and then
    3. with the smaller internal rule ID in the sequence

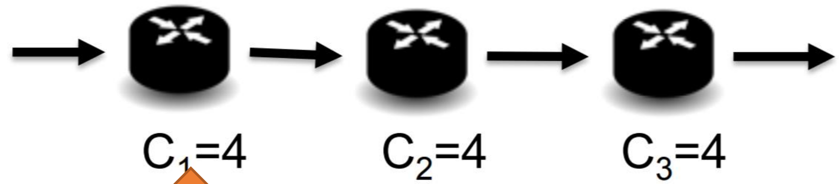




R2

X

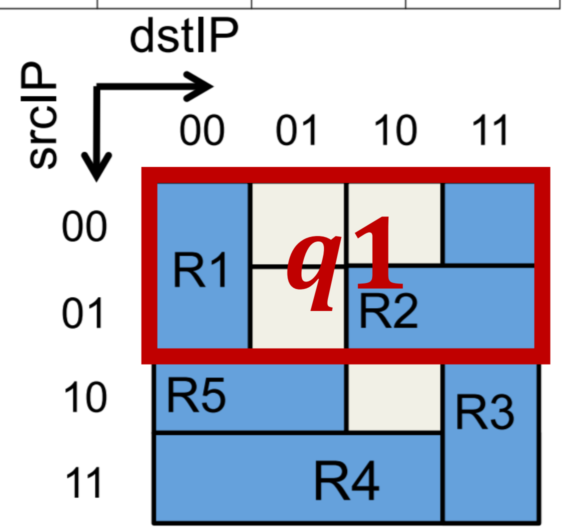
R5 ***X***



Example

- Find the rectangle with the maximum utility

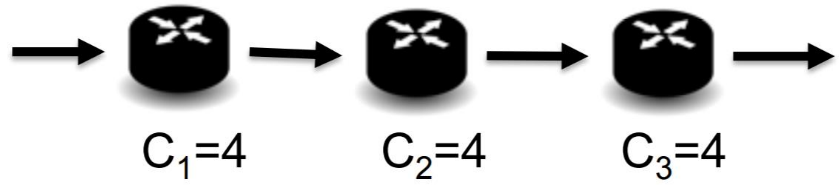
Rule	Source IP	Destination IP	Action
R1	0*	00	Permit
R2	01	1*	Permit
R3	*	11	Drop
R4	11	*	Permit
R5	10	0*	Permit
R6	*	*	Drop



Rule	src	dst	action
r1	0*	00	Permit
r2	01	1*	Permit
r3	0*	11	Drop
r4	0*	*	Drop

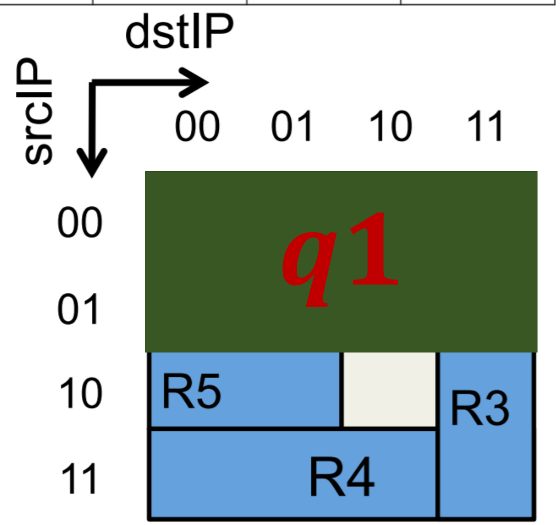
$$\boldsymbol{r_1: (q1 \wedge R1.p, R1.a)}$$
$$\boldsymbol{r_2: (q1 \wedge R2.p, R2.a)}$$
$$\boldsymbol{r_3: (q1 \wedge R3.p, R3.a)}$$
$$\boldsymbol{r_4: (q1 \wedge R6.p, R6.a)}$$

Example



- Rewrite the remaining rules

Rule	Source IP	Destination IP	Action
R1	0*	00	Permit
R2	01	1*	Permit
R3	*	11	Drop
R4	11	*	Permit
R5	10	0*	Permit
R6	*	*	Drop



Rule	src	dst	action
r1	0*	*	Fwd
r2	*	11	Drop
r3	11	*	Permit
r4	10	0*	Permit
r5	*	*	Drop

$r_1: (q1, Fwd)$

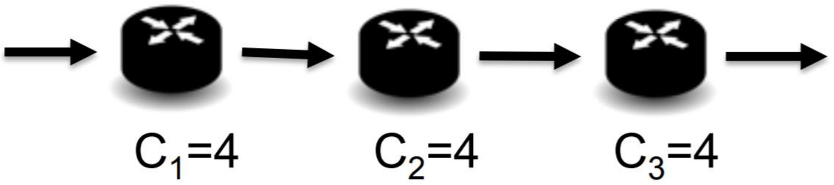
$r_2: (R3.p, R3.a)$

$r_3: (R4.p, R4.a)$

$r_4: (R5.p, R5.a)$

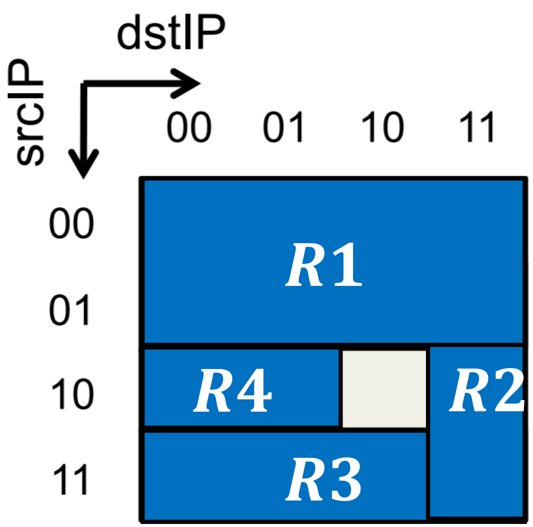
$r_5: (R6.p, R6.a)$

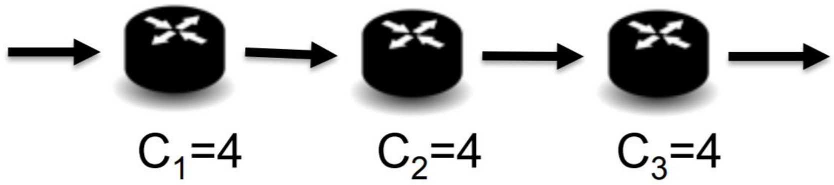
Example



- Regard the remaining rules as a new set of rules

Rule	src	dst	action
R1	0*	*	Fwd
R2	*	11	Drop
R3	11	*	Permit
R4	10	0*	Permit
R5	*	*	Drop

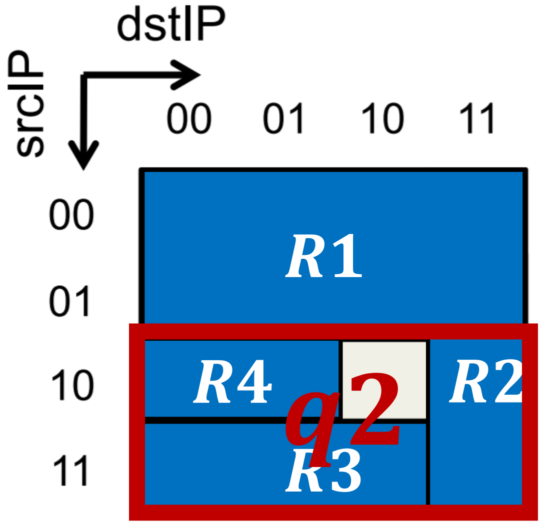


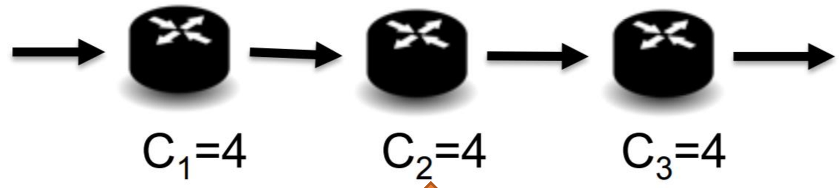


Example

- Find the rectangle with the maximum utility

$$\textit{utility}(q_2) = \frac{2-1}{2+1} = \frac{1}{3}$$



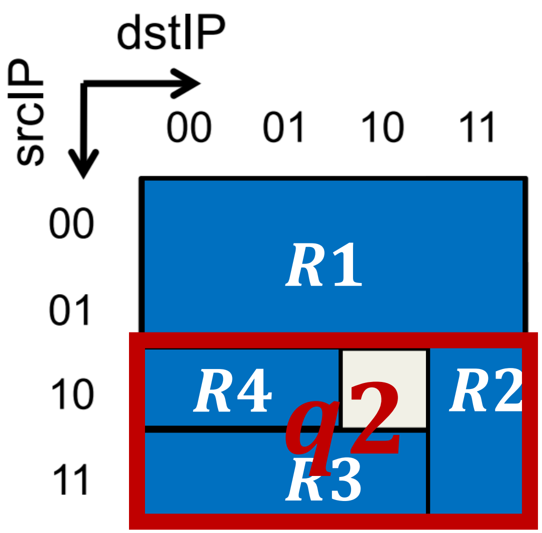


Example

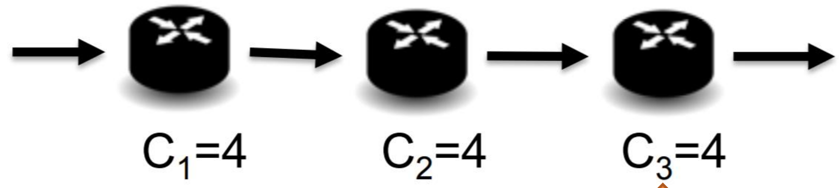
- Find the rectangle with the maximum u lity

Rule	src	dst	action
R1	0*	*	Fwd
R2	*	11	Drop
R3	11	*	Permit
R4	10	0*	Permit
R5	*	*	Drop

Rule	src	dst	action
r1	1*	11	Drop
r2	11	*	Permit
r3	10	0*	Permit
r4	1*	*	Drop



$$r_1: (q2 \wedge R2.p, R2.a)$$
$$r_2: (q2 \wedge R3.p, R3.a)$$
$$r_3: (q2 \wedge R4.p, R4.a)$$
$$r_4: (q2 \wedge R5.p, R5.a)$$

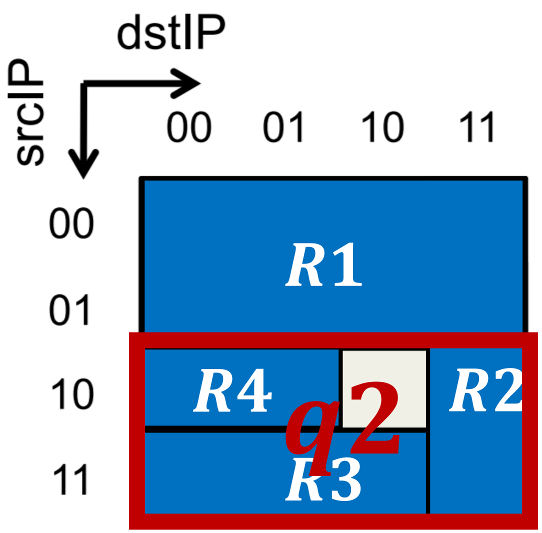


Example

- Rewrite the remaining rules

Rule	src	dst	action
R1	0*	*	Fwd
R2	*	11	Drop
R3	11	*	Permit
R4	10	0*	Permit
R5	*	*	Drop

Rule	src	dst	action
r1	1*	*	Fwd
r2	0*	*	Fwd



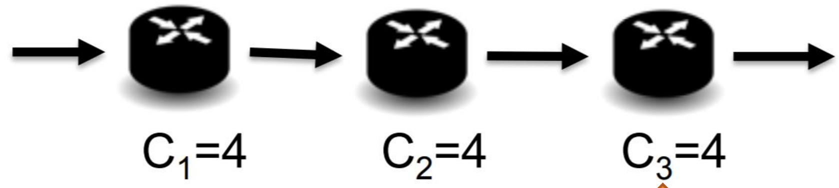
r_3 and r_4 are removed because they are fully covered by r_1 and r_2

$r_1: (q2, Fwd)$

$r_2: (R1.p, R1.a)$

$\cancel{r_3: (R2.p, R2.a)}$

$\cancel{r_4: (R5.p, R5.a)}$

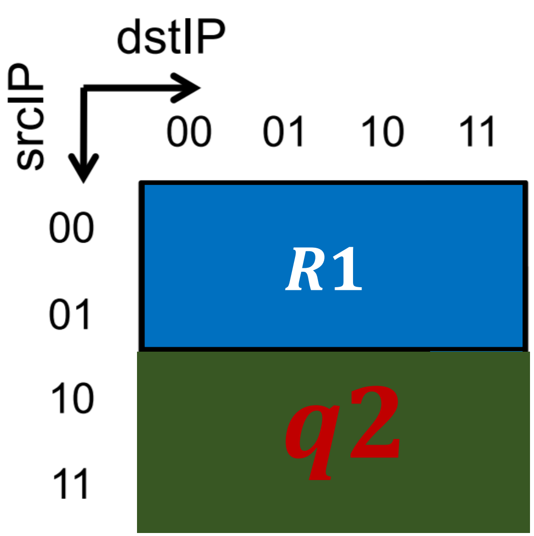


Example

- Rewrite the remaining rules

Rule	src	dst	action
R1	0*	*	Fwd
R2	*	11	Drop
R3	11	*	Permit
R4	10	0*	Permit
R5	*	*	Drop

Rule	src	dst	action
r1	1*	*	Fwd
r2	0*	*	Fwd



$\boldsymbol{r_3}$ and $\boldsymbol{r_4}$ are removed because they are fully covered by $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$

$$r_1: (q2, Fwd)$$
$$r_2: (R1.p, R1.a)$$

Discussion

- The related problems are NP-hard
- You cannot find an efficient optimal algorithm for these problems unless NP = P

- There are many heuristic algorithms
  - "Too many SDN rules? Compress them with MINNIE," in IEEE ICC 2015
  - "Optimizing the One Big Switch Abstraction in Software-Defined Networks," in ACM CoNext 2013
  - "Palette: Distributing tables in software-defined networks," in IEEE INFOCOM 2013

Input Sample: input.txt

Format:

#Bits
IngressID1 EgressID2
#switches
switchID capacity
#links
switchID1 switchID2 weight
...
#rules
SrcIP1 DstIP2 action
...

Rule	Source IP	Destination IP	Action
R1	0*	00	Permit
R2	01	1*	Permit
R3	*	11	Drop
R4	11	*	Permit
R5	10	0*	Permit
R6	*	*	Drop

Input Sample: input.txt

```
2
0    2
3
0         4
1         4
2         4
2
0    1    1
1    2    1
6
0*        00        Permit
01        1*        Permit
*         11        Drop
11        *         Permit
10        0*        Permit
*         *         Drop
```

$C_0 = 4$ $C_1 = 4$ $C_2 = 4$

Rule	Source IP	Destination IP	Action
R1	0*	00	Permit
R2	01	1*	Permit
R3	*	11	Drop
R4	11	*	Permit
R5	10	0*	Permit
R6	*	*	Drop

Output Sample: output.txt

Format:

#tables(#switchesOfPath)
switchID1 #entriesOfTable1
SrcIP1 DstIP2 action
...
switchID2 #entriesOfTable2
SrcIP1 DstIP2 action
...

Rule	src	dst	action
r1	0*	00	Permit
r2	01	1*	Permit
r3	0*	11	Drop
r4	0*	*	Drop

Rule	src	dst	action
r1	1*	11	Drop
r2	11	*	Permit
r3	10	0*	Permit
r4	1*	*	Drop

Rule	src	dst	action
r1	1*	*	Fwd
r2	0*	*	Fwd

PS: You need to return all the tables of switches on the path even if the number of entries of the table is empty

Output Sample: output.txt

```
3
0    4
0*   00   Permit
01   1*   Permit
0*   11   Drop
0*   *    Drop
1    4
1*   11   Drop
11   *    Permit
10   0*   Permit
1*   *    Drop
2    2
1*   *    Fwd
0*   *    Fwd
```

Rule	src	dst	action
r1	0*	00	Permit
r2	01	1*	Permit
r3	0*	11	Drop
r4	0*	*	Drop

Rule	src	dst	action
r1	1*	11	Drop
r2	11	*	Permit
r3	10	0*	Permit
r4	1*	*	Drop

Rule	src	dst	action
r1	1*	*	Fwd
r2	0*	*	Fwd

Note

- Superb deadline: 12/3 Thu (視情況延期)
- Deadline: 12/10 Thu (視情況延期)
- Submit your code to E-course2
- Demonstrate your code in 工院1館 401B
- **C Source code**
- Show a good programming style

Appendix: Entry Utilization Minimization
Non-prefix Compression

- The original table → 9 entries
- The modified table by wildcard (*,n) → 6 entries

Flow	Output port
(0, 4)	Port-4
(0, 5)	Port-5
(0, 6)	Port-5
(1, 4)	Port-6
(1, 5)	Port-4
(1, 6)	Port-6
(2, 4)	Port-4
(2, 5)	Port-5
(2, 6)	Port-6

(a) Without Compression

Flow	Output port
(1, 4)	Port-6
(1, 5)	Port-4
(0, 6)	Port-5
$(*, 4)$	Port-4
$(*, 5)$	Port-5
$(*, *)$	Port-6

(d) With $(*, n)$ rule

Appendix: Entry Utilization Minimization Non-prefix Compression

- The original table → 9 entries
- The minimal table → 5 entries

Flow	Output port
(0, 4)	Port-4
(0, 5)	Port-5
(0, 6)	Port-5
(1, 4)	Port-6
(1, 5)	Port-4
(1, 6)	Port-6
(2, 4)	Port-4
(2, 5)	Port-5
(2, 6)	Port-6

(a) Without Compression

Flow	Output port
(1, 5)	Port-4
(2, 6)	Port-6
$(1, *)$	Port-6
$(*, 4)$	Port-4
$(*, *)$	Port-5

(e) Minimal solution