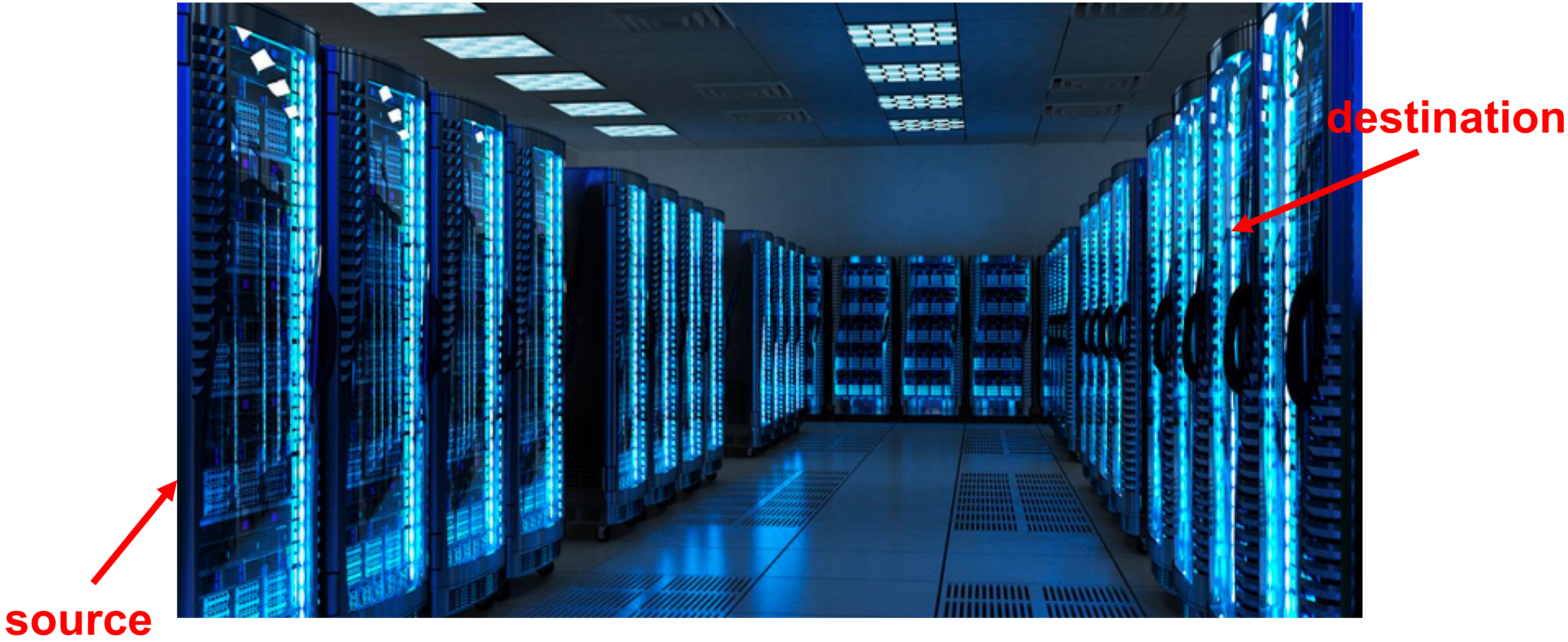
Object-Oriented Programming
Programming Project #1

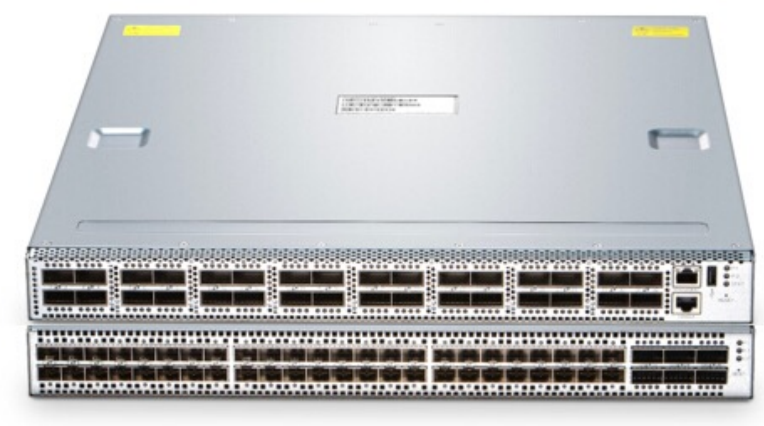
Data Center

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



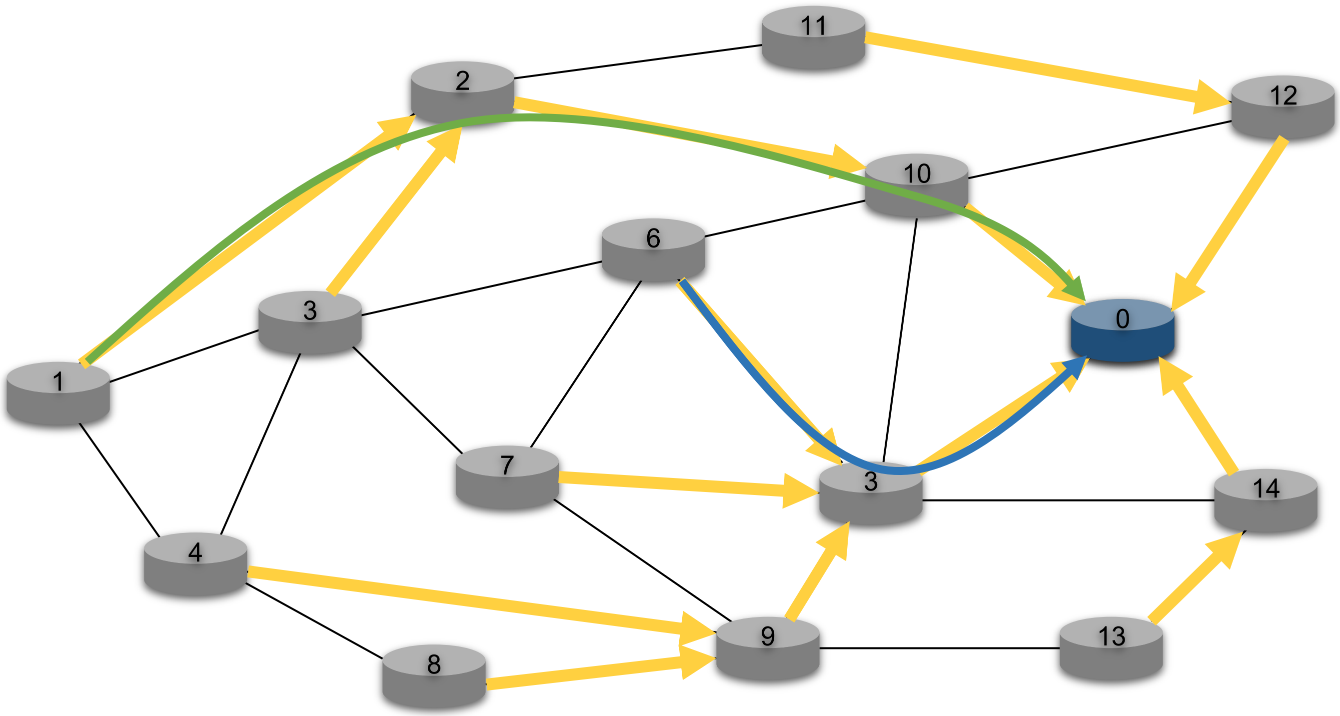
Switches

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



Traditional Routing Path

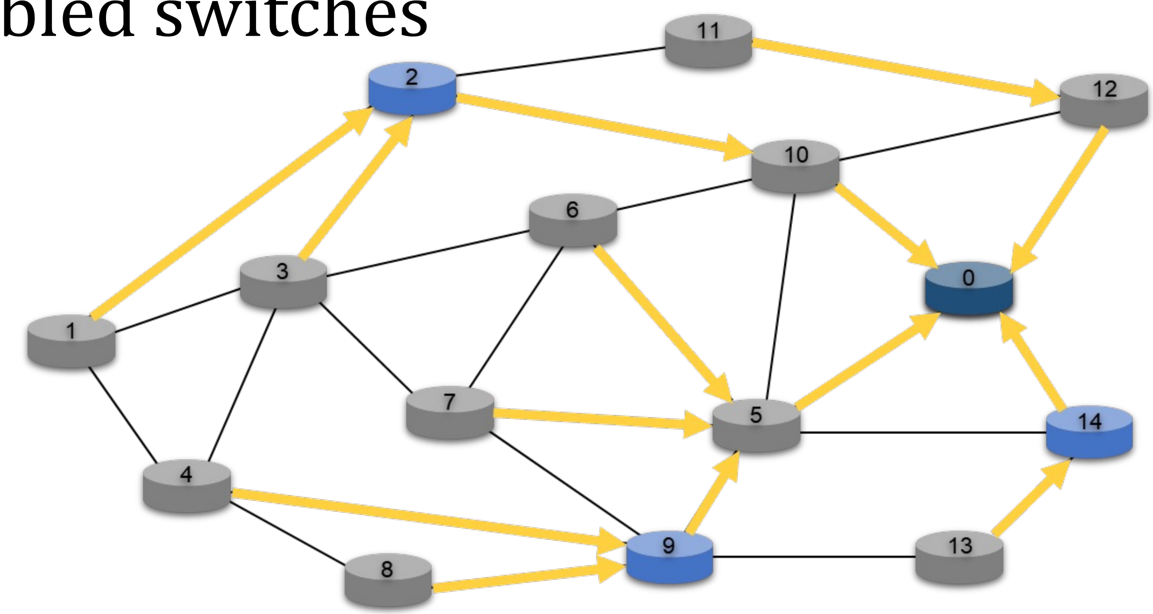
- Switches use OSPF (i.e., shortest path)
- Construct a shortest path tree rooted at each destination



Disadvantages of OSPF Routing Tables

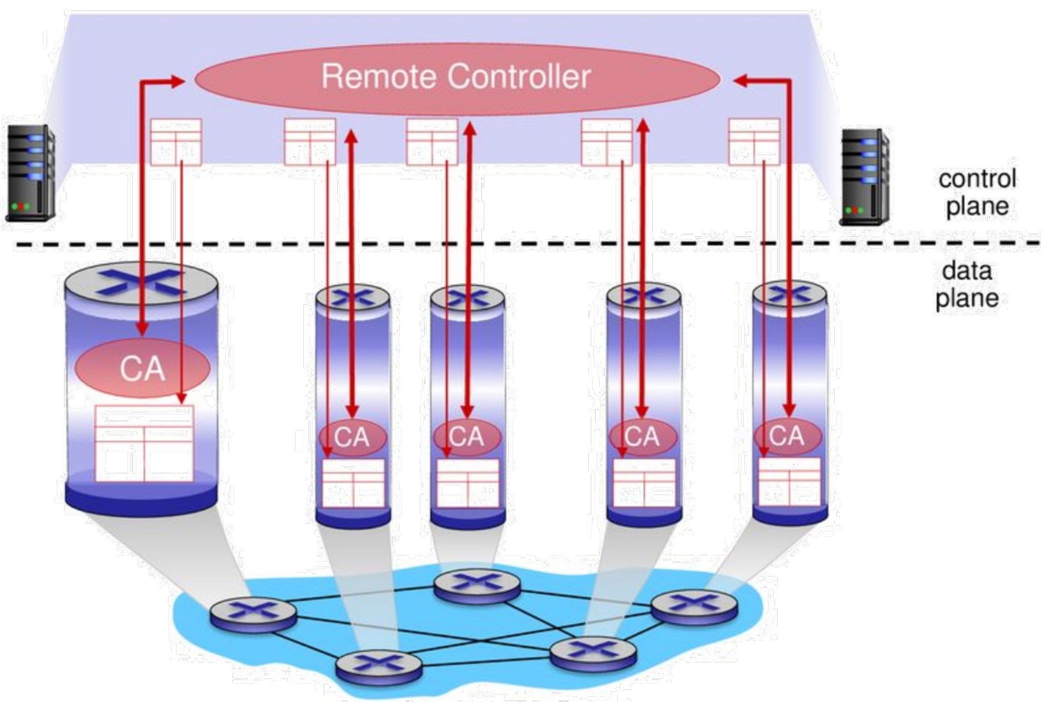
- All paths are fixed → Not flexible
- Periodical update → Not real-time

- We need SDN-enabled switches

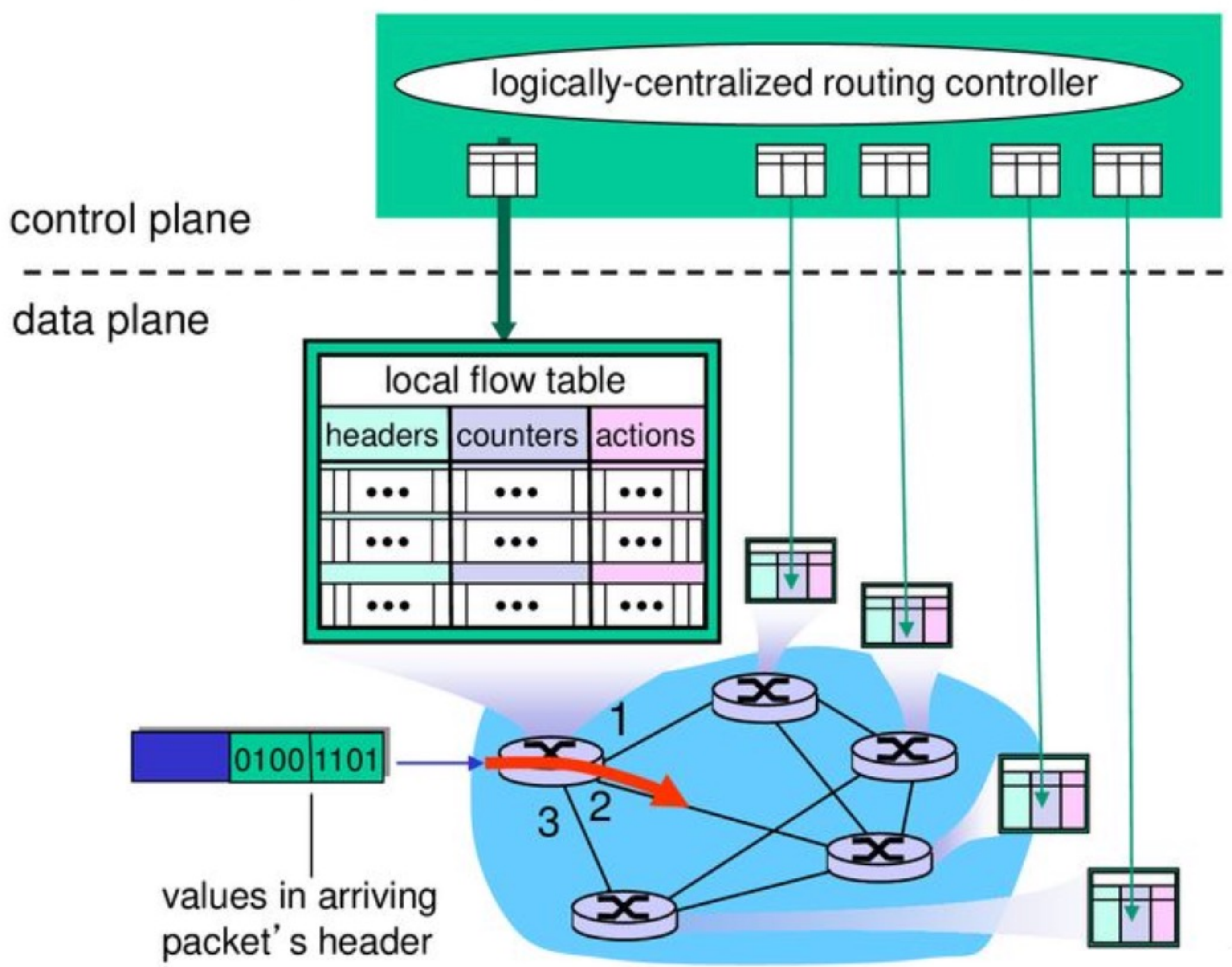


SDN-enabled Switches

- A centralized controller is introduced – software-defined networking (SDN)

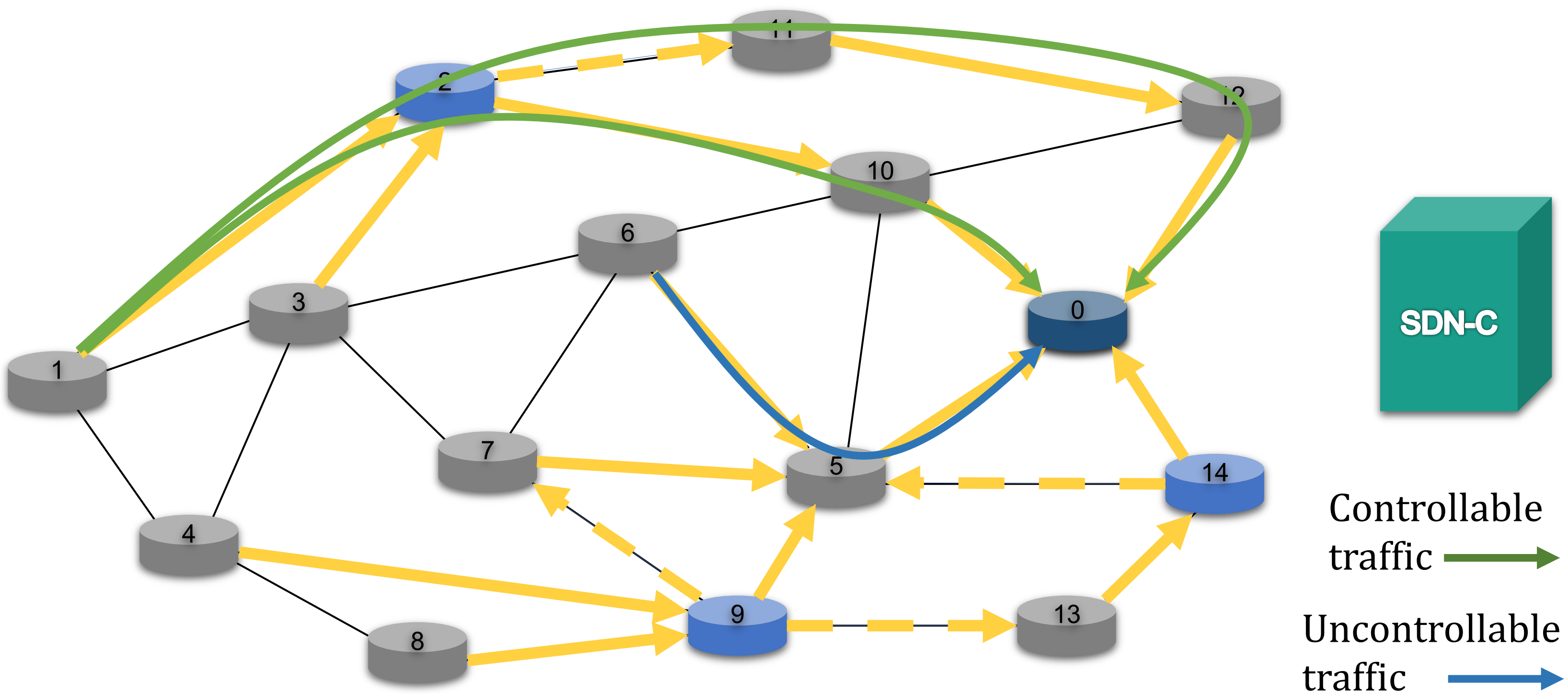


Installing Rules in the SDN-enabled Switches



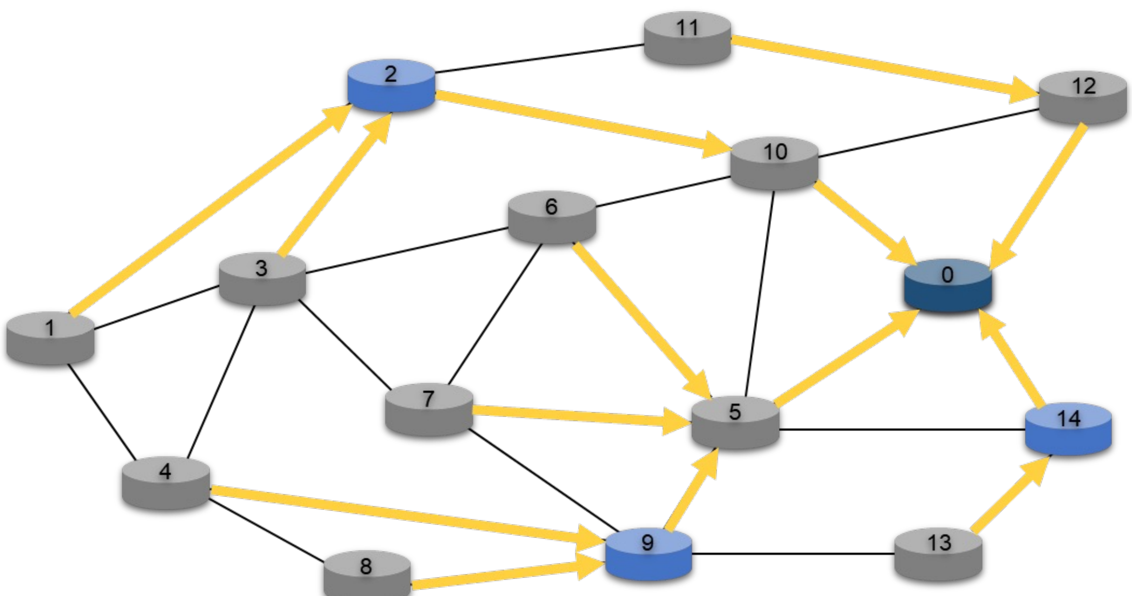
Incrementally Deployed SDN

- Non-SDN switches: uncontrollable OSPF paths
- SDN switches: controllable paths



OSPF Routing Information

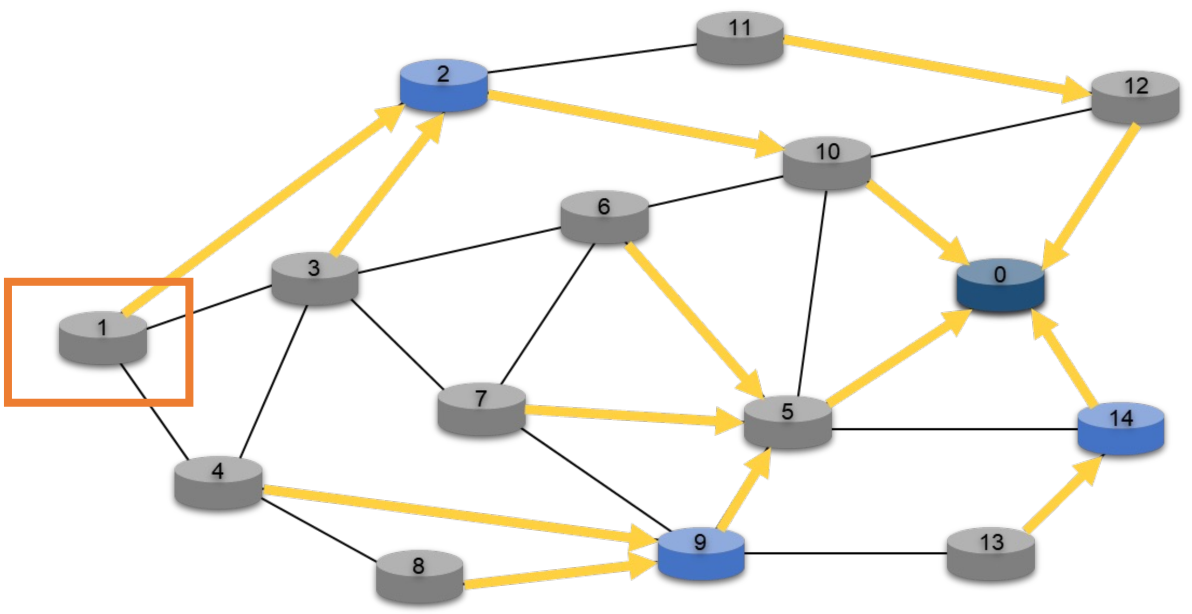
- Given: a graph with links and destinations
- Output: shortest paths towards all destinations
- Then, store the information in each node’s table



OSPF Routing Table

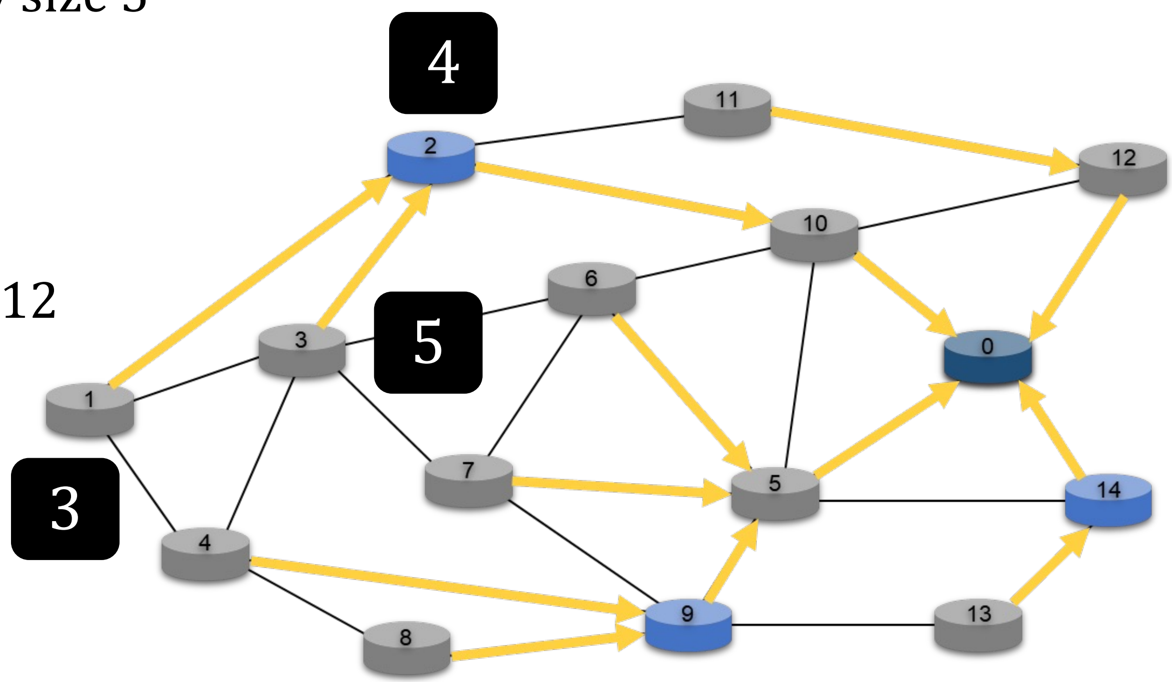
- Key: each destination
- Value: the next node (i.e., the output port)
- Node 1's table (it uses OSPF)

Destination	Next Node
0	2



Routing Flows with OSPF Routing Tables

- Given flows:
  - Flow 1 → 0 with flow size 3
  - Flow 2 → 0 with flow size 4
  - Flow 3 → 0 with flow size 5
- Links' loads:
  - Link (1, 2)'s load = 3
  - Link (3, 2)'s load = 5
  - Link (2, 10)'s load = 12
- Max link load = max{3, 5, 12} = 12

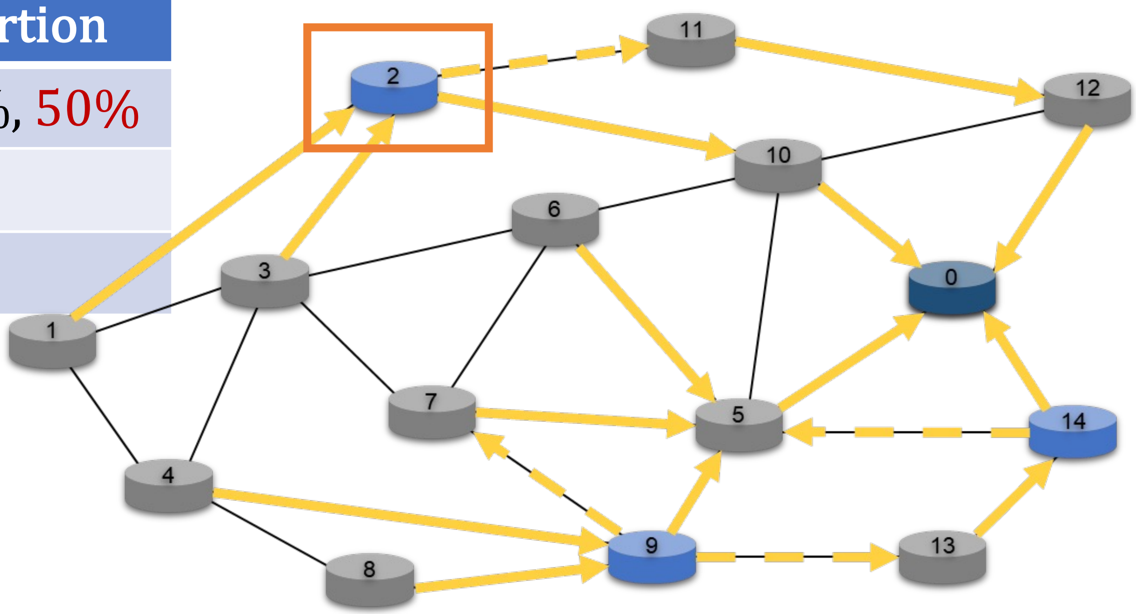


SDN-enabled Routing Table

- Key: each destination
- Value: the next nodes (i.e., the output ports)
- Node 2's table (it is SDN-enabled)

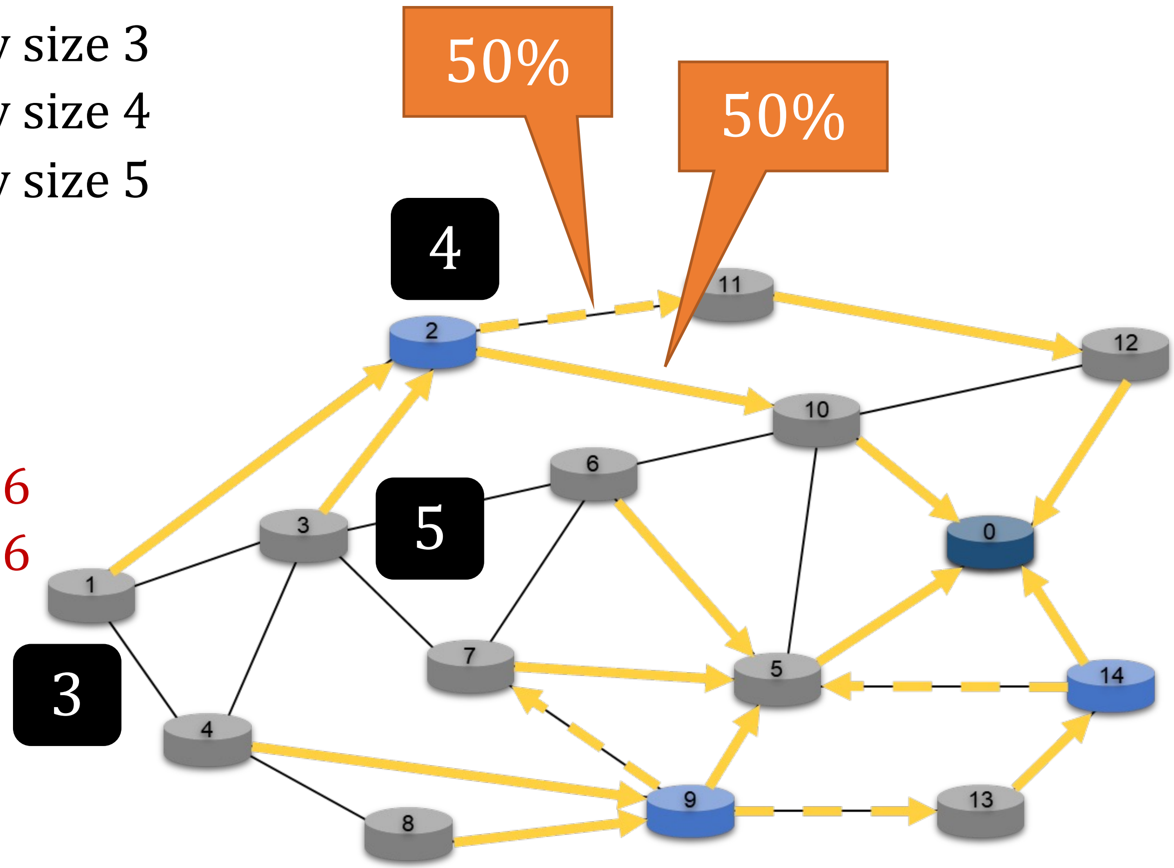
Destination	Next Node	Portion
0	10, 11	50%, 50%

The controller can define the portions



Routing Flows with Hybrid Routing Tables

- Given flows:
  - Flow 1 → 0 with flow size 3
  - Flow 2 → 0 with flow size 4
  - Flow 3 → 0 with flow size 5
- Links' loads:
  - Link (1, 2)'s load = 3
  - Link (3, 2)'s load = 5
  - Link (2, 10)'s load = 6
  - Link (2, 11)'s load = 6
- Max link load
  = max {3, 5, 6, 6}
  = 6



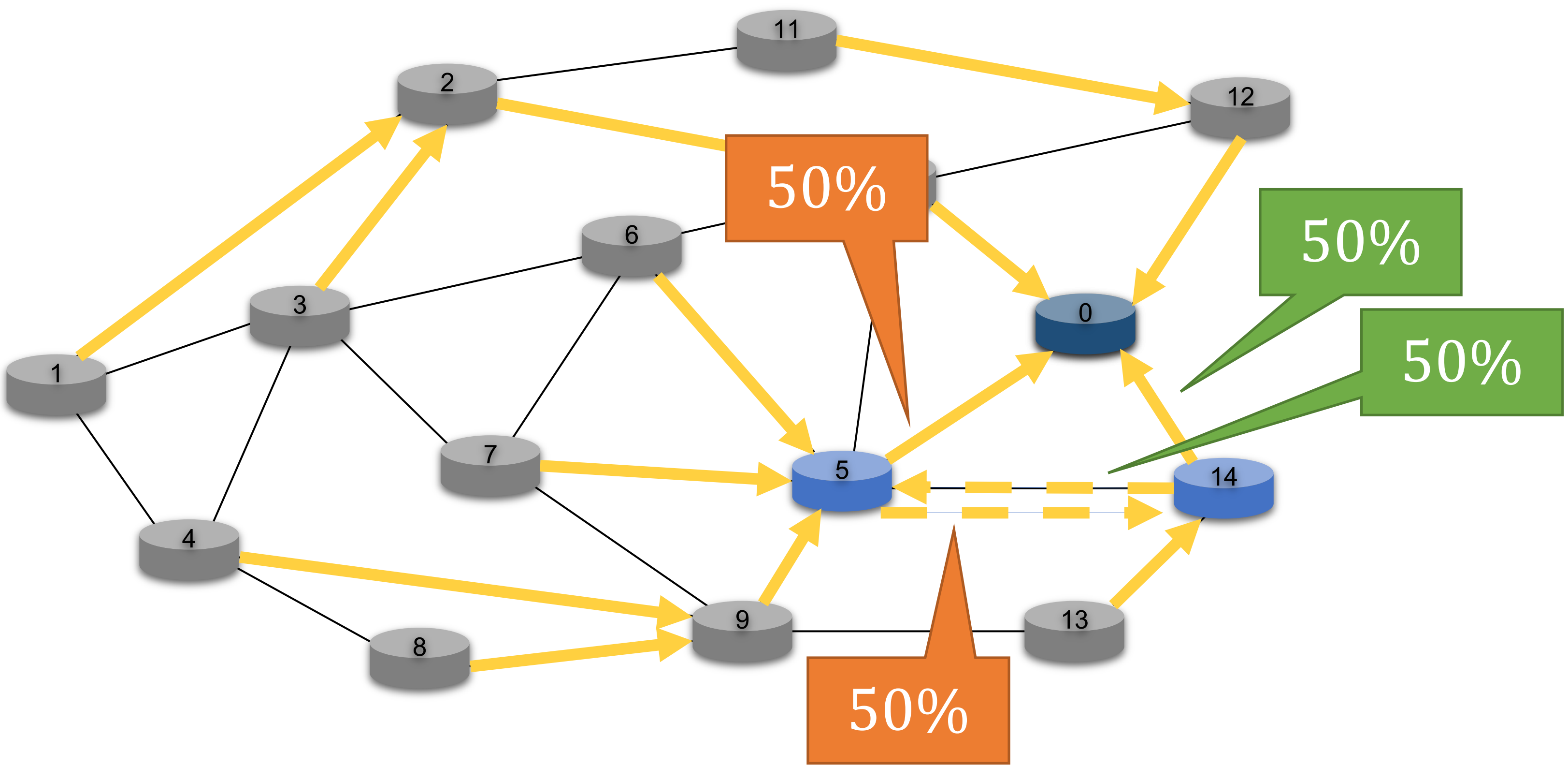
Requirements

- Every node only knows its neighbors
- Define your own node class and use public

  (i.e., just use class instead of struct)
- Each node has an unsigned int ID
- Use a vector<unsigned int> to store the neighbors' IDs in each node
- Use a map<unsigned int, vector<pair<unsigned int, double> > > to store each entry in the table (i.e., each entry in the table has destination ID, <next nodes' IDs, portions>)

Note

- Serious congestion problems happen if cycles exist
- Avoid cycles in the routing paths for each destination



Programming Project #1:
Routing Table in an Incrementally Deployed SDN

- Input:
  - # nodes, #SDN nodes, # destinations, # links, and #pairs
  - SDN nodes (ID)
  - Destinations (ID)
  - Links between nodes
  - Traffic matrix (flow size for each pair)
- Procedure:
  - Compute shortest paths to each destination
  - Compute next hops and portions for SDN-enabled nodes
- Output:
  - Each node's routing table

The Competition

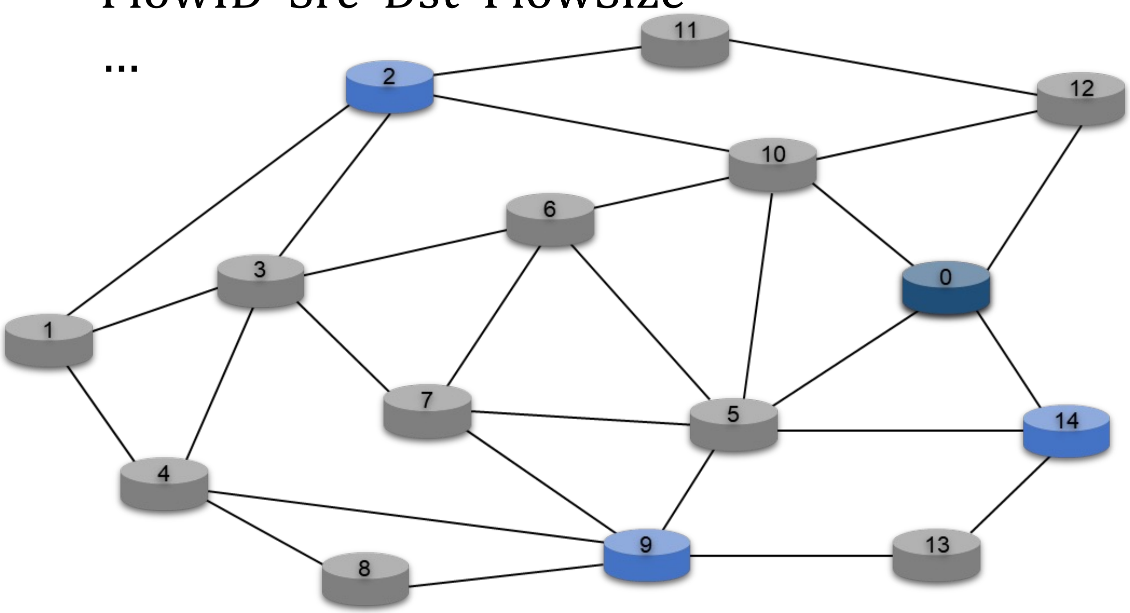
- The grade is inversely proportional to the max link load
- Basic: 60 (deadline)
  - Every node's packet can be sent to the destination with no cycle
- Being a coding assistant (superb deadline)
  - +10
- Performance ranking (decided after the deadline)
  - [0%, 30%) (bottom): +0
  - [30%, 50%): + 5
  - [50%, 75%): + 10
  - [75%, 85%): + 15
  - [85%, 90%): + 20
  - [90%, 95%): + 25
  - [95%, 100%] (top): + 30

The Competition

- Note that you cannot use brute-force algorithm
- Note that your code must be deterministic (no randomization)

Input Sample:
use cin

Format:
#Nodes #SDN_Nodes #Dsts #Links #Pairs
SDN_NodeID_List
DstID_List
...
LinkID Node1 Node2
...
FlowID Src Dst FlowSize
...



15 3 1 28 3
2 9 14
0
0 0 5
1 0 10
2 0 12
3 0 14
4 1 2
5 1 3
6 1 4
7 2 3
8 2 10
9 2 11
10 3 4
11 3 6
12 3 7
13 4 8
14 4 9
15 5 6
16 5 7
17 5 9
18 5 10
19 5 14
20 6 7
21 6 10
22 7 9
23 8 9
24 9 13
25 10 12
26 11 12
27 13 14
0 1 0 3
1 2 0 4
2 3 0 5

Output Sample (not optimal):

use cout

Format:

NodeID

DstID NextID

...

e.g.,

0

0 0

Its own ID

1

0 2

2

0 10 50% 11 50%

3

0 2

4

0 9

5

0 0

6

0 5

7

0 5

8

0 9

9

0 5 60% 7 0% 13 40%

10

0 0

11

0 12

12

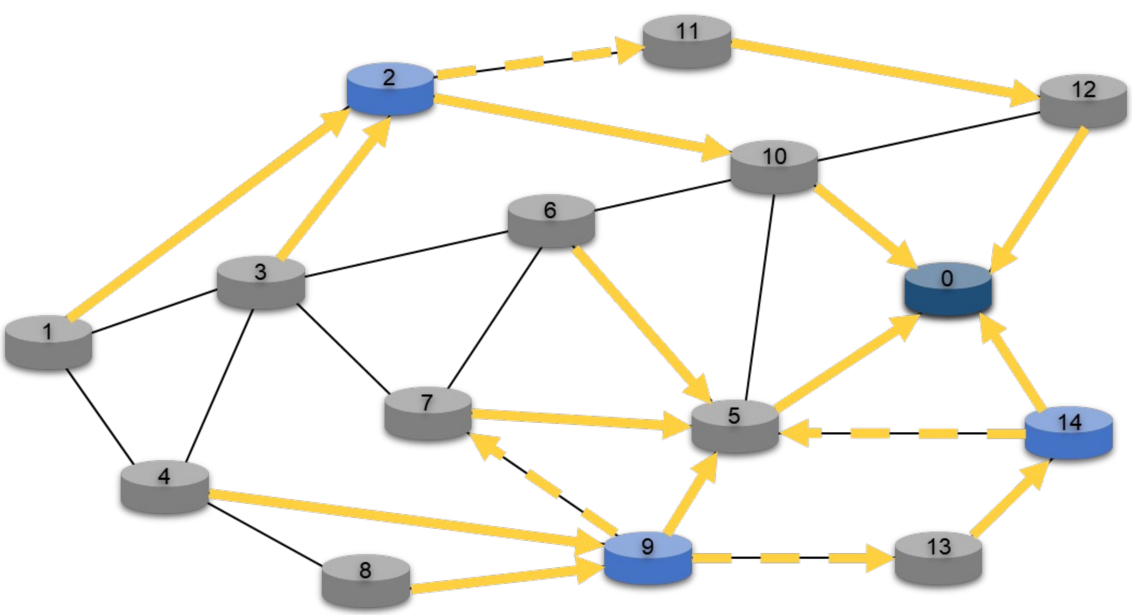
0 0

13

0 14

14

0 0 70% 5 30%



Note

- Superb deadline: 3/16 Thu
- Deadline: 3/23 Thu
- Pass the test of our online judge platform
- Submit your code to E-course2
    - The file name should be ``OOP_HW1_studentID.cpp''
- Demonstrate your code remotely with TA
- **C++ Source code (only C++; compiled with g++)**
    - **Include C++ library only (i.e., no stdio, no stdlib, ...)**
    - **Please use new and delete instead of malloc and free**
- Show a good programming style