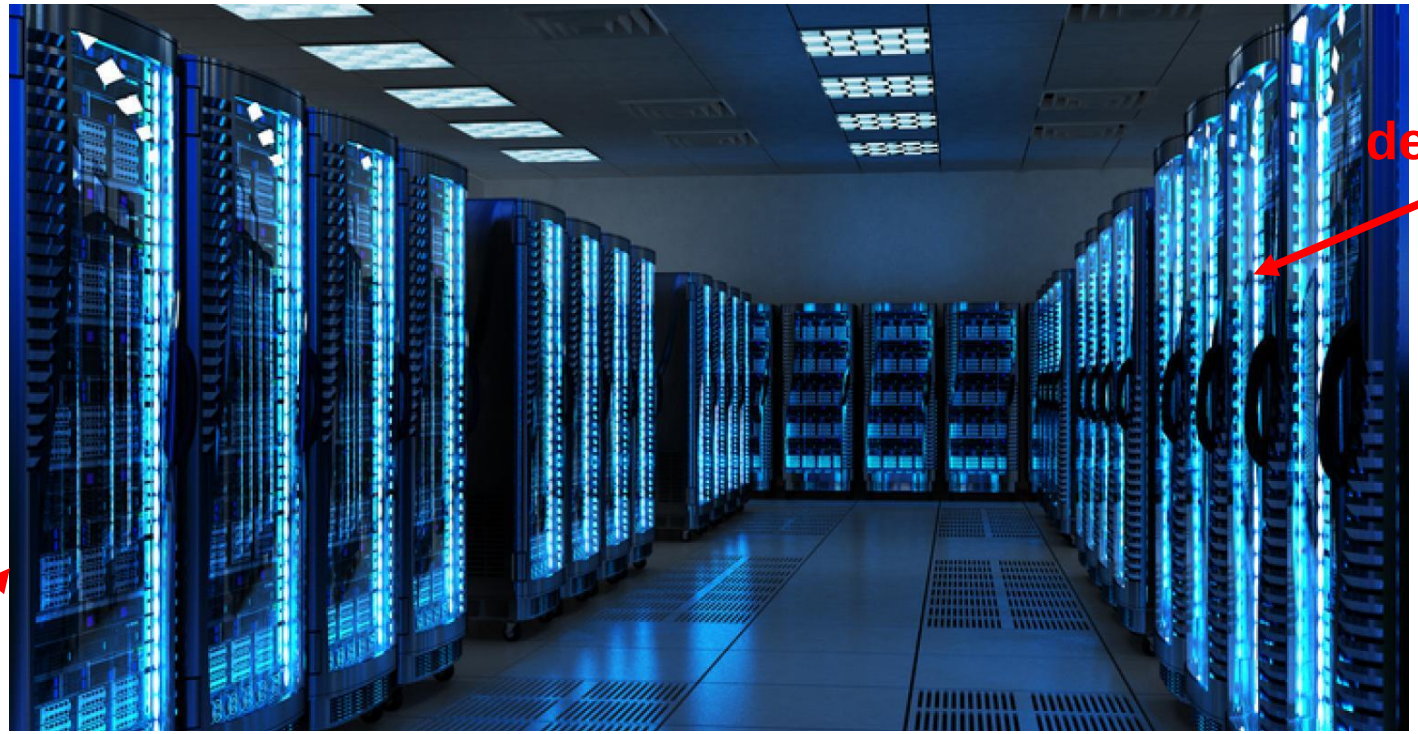
Object-Oriented Programming
Programming Project #2

Data Center

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

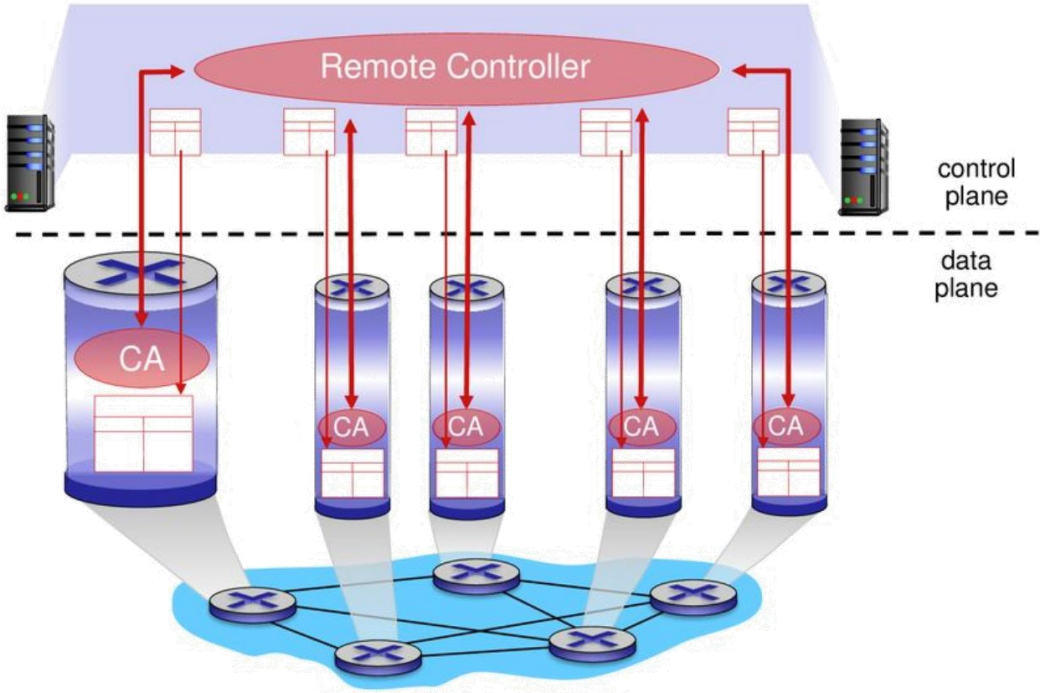


Switches

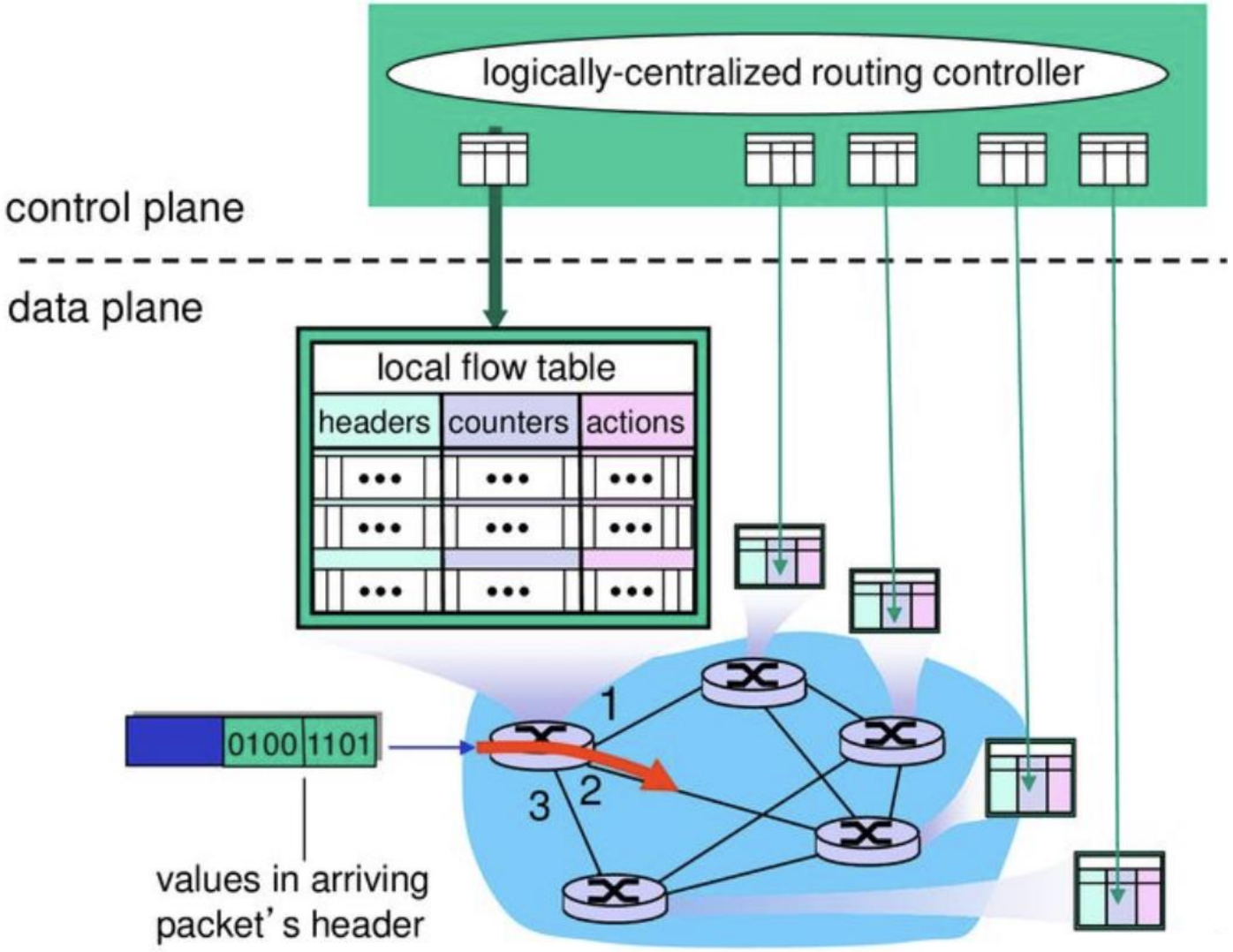
- Each switch has multiple ports
- Receive and forward the packets from a port to another port

SDN-enabled Switches

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

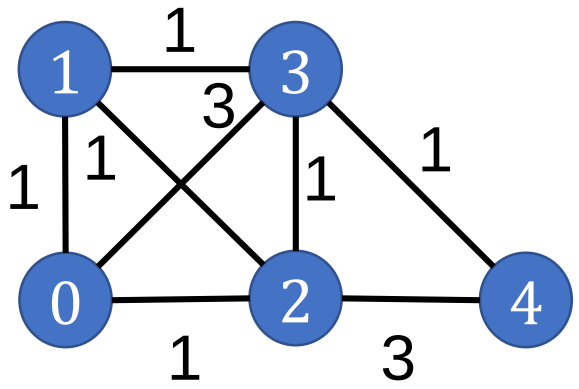


Installing Rules in the SDN-enabled Switches

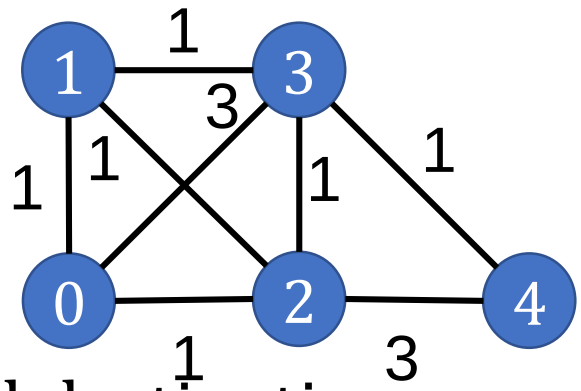


Routing Information

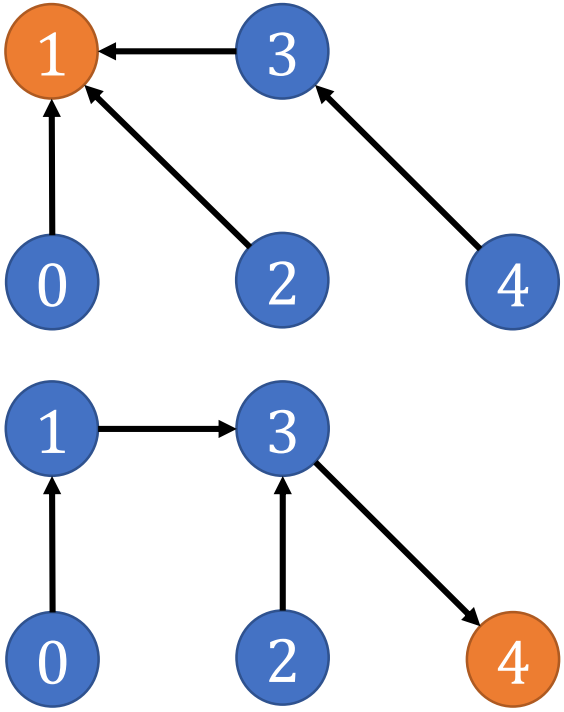
- Input: a graph with link weights and destinations
- Output: shortest paths towards all destinations
  - (Tie breaking) If two next nodes have the same hop, then choose the one with a smaller ID
- Then, store the information in each node's table



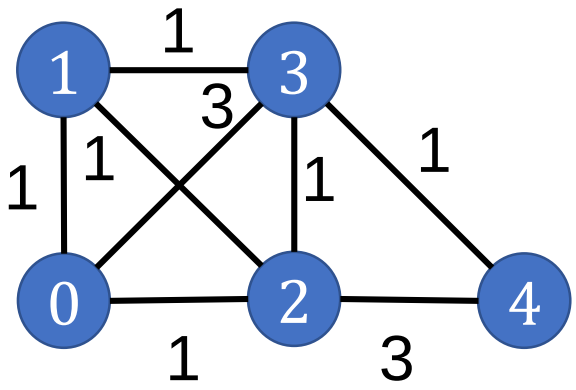
Routing Paths (Trees)



- Input: a graph with link weights and destinations
- Output: shortest paths towards all destinations
- Shortest path trees (SPTs)

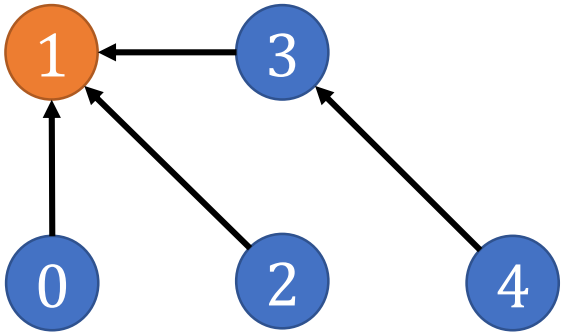


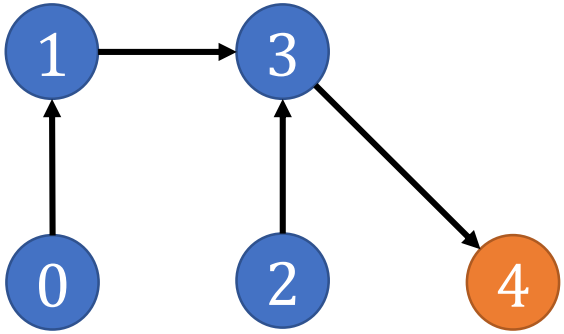
Routing Table



- Key: each destination
- Value: the next node (i.e., the output port)
- Node 0's table

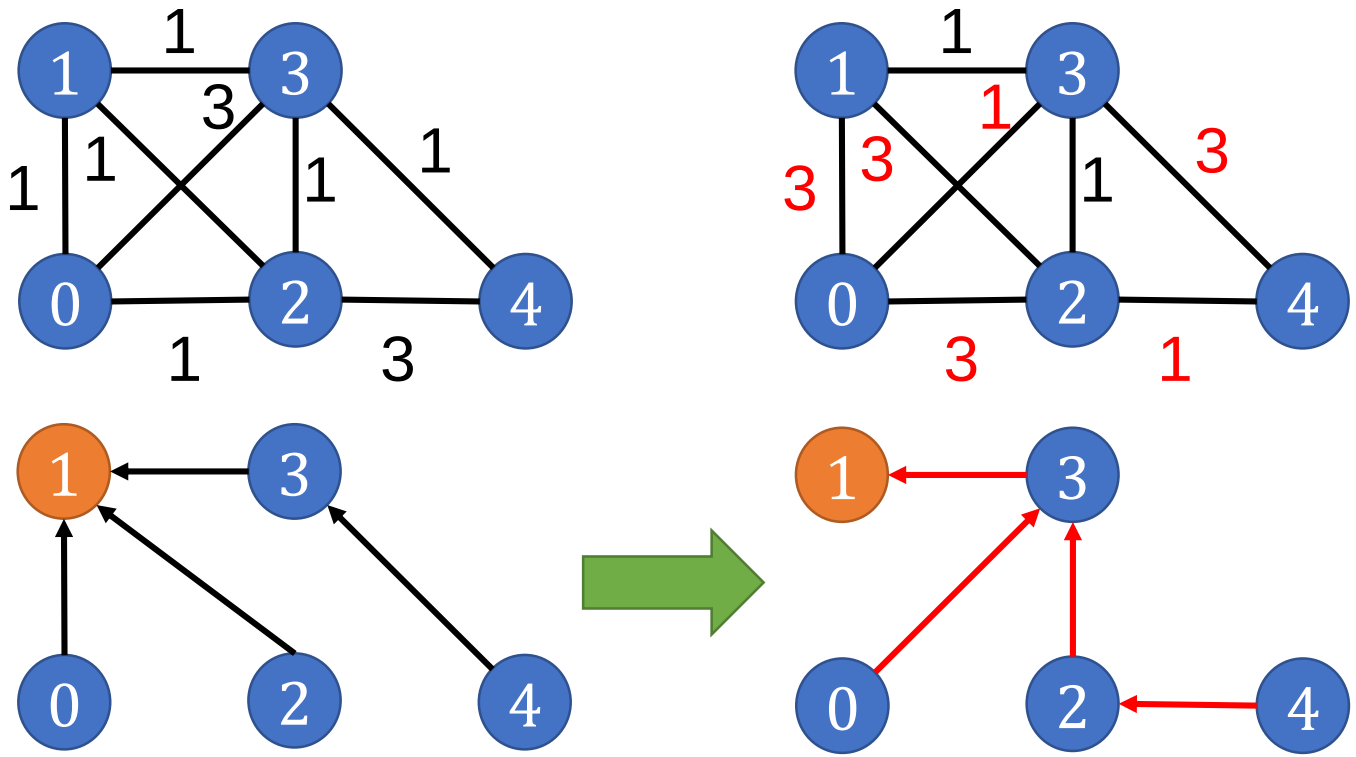
Destination	Next Node
1	1
4	1





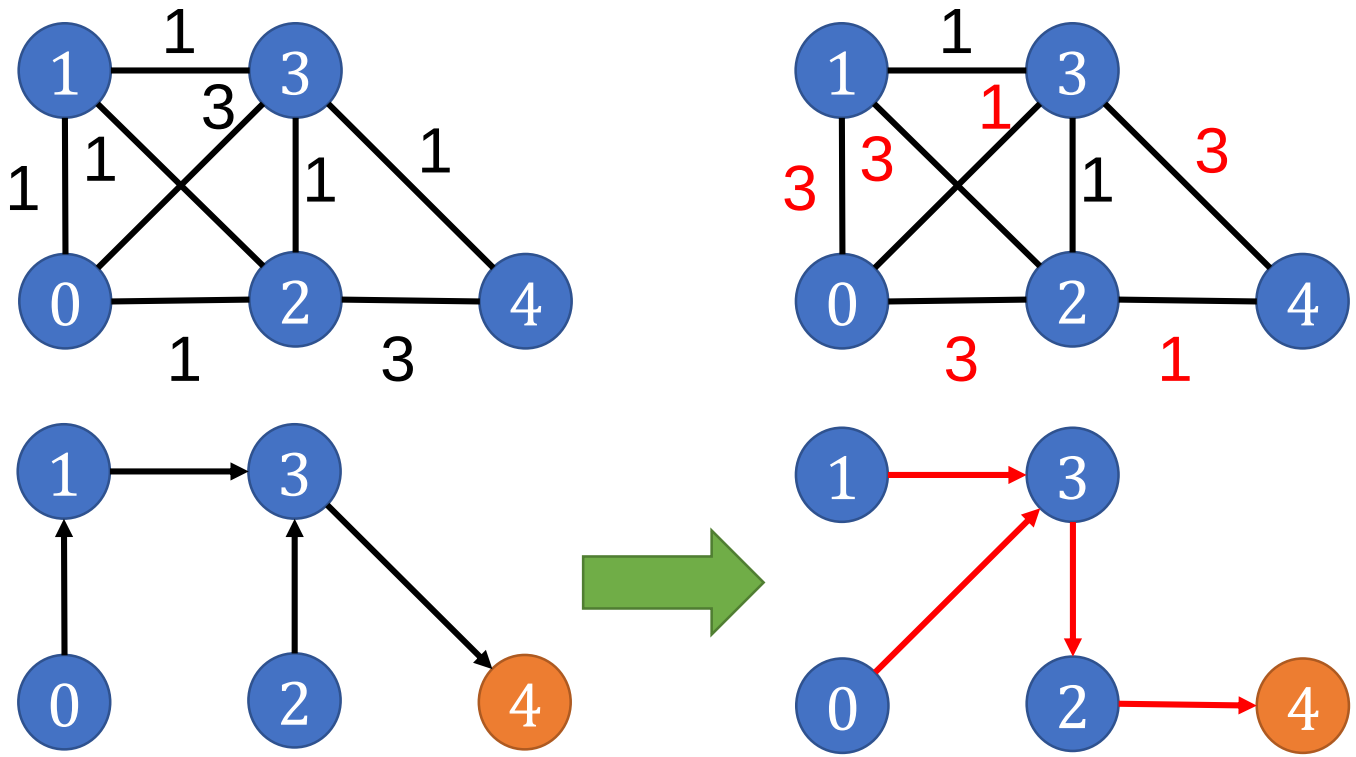
Routing Path Update (aka Network Update)

- Input: the same graph with new link weights
- Output: the new next node (i.e., the output port)



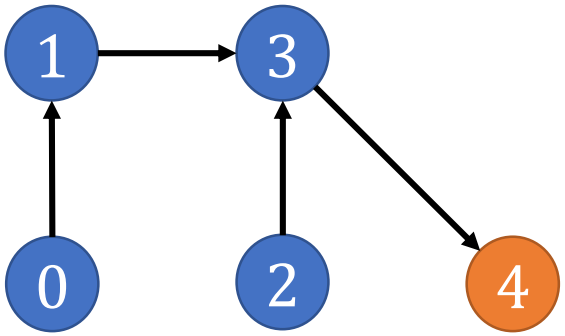
Routing Path Update (aka Network Update)

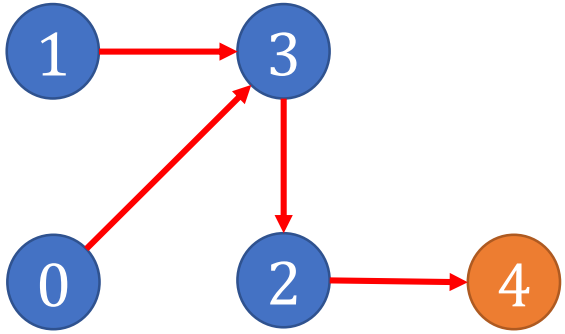
- Input: the same graph with new link weights
- Output: the new next node (i.e., the output port)



Routing Path Update (aka Network Update)

- For each destination, a new tree is generated
- The information in the table should be updated





Node ID	0	1	2	3	4
To 4	1	3	3	4	-1

Node ID	0	1	2	3	4
To 4	3	3	4	2	-1

Note

- Create switches with class SDN_switch
- Each switch has an unsigned int ID
  - node::node_generator::generate("SDN_switch",id);
- Every node only knows its neighbors
- Add the neighbors for each switch
  - node::id_to_node(0)->add_phy_neighbor(1);
  - node::id_to_node(1)->add_phy_neighbor(0);
  - We use simple_link with a fixed latency (i.e., 10)
- Write a map<unsigned int, unsigned int> to store each entry in each switch's table (i.e., <destination ID, next node ID>) in class SDN_switch
  - Copy and modify the routing table code in HW1

Note

- Define the rules to handle the received packet in class SDN_swtich's member function recv_handler
  - void SDN_switch::recv_handler (packet *p)
  - Don't use node::id_to_node(id) in recv_handler
- Get the current switch's ID and its neighbor
  - Use getNodeID() in recv_handler
  - Use getPhyNeighbors().find(n_id) to check whether the node with n_id is a neighbor
  - Use const map<unsigned int,bool> &nblist =getPhyNeighbors() and for (map<unsigned int,bool>::const_iterator it = nblist.begin(); it != nblist.end(); it ++) to get all neighbors
- Use send_handler(packet *p) to send the packet
- Check the packet type
  - if (p->type() == "SDN_data_packet")

Note

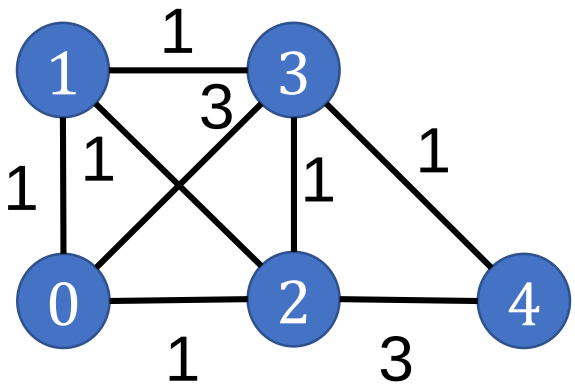
- Decode: Cast the packet to the right type
  - dynamic_cast<SDN_data_packet *> (p)
  - dynamic_cast<SDN_ctrl_packet *> (p)
  - …
  - Will be explained in the later chapters
- Generate the data and control packets
  - void data_packet_event(unsigned int src, unsigned int dst, unsigned int t, string msg)
  - A packet will be sent from a source to a destination at time t
  - void ctrl_packet_event(unsigned int dst, unsigned int t, string msg)
  - A packet received by a switch at time t with the content msg
- The control packet's msg is used to update an entry in the switch's routing table
  - A packet received by a switch at time t with the content msg
  - msg contains: "dstID nexID"

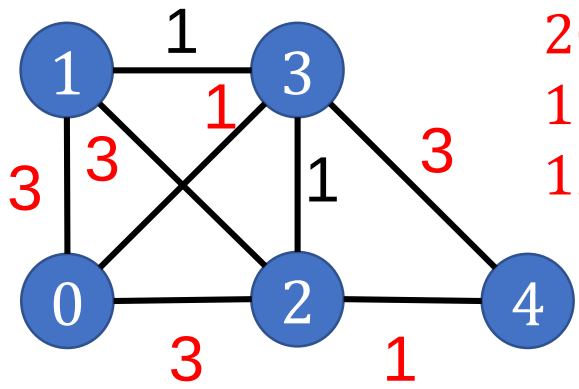
Programming Project #1: Routing Table

- Input:
  - Numbers of nodes, destinations, and links
  - Destinations
  - Links with **old** weights
  - Links with new weights
  - Source-destination pairs with forward start time
  - Network update time (1^{st} and 2^{nd} are for installation and update, resp.)
  - Simulation duration
- Procedure:
  - Compute **old** shortest paths to destinations
  - Compute new shortest paths to destinations
  - Forward the packet from the source to the destination at the forward start time
  - Install and update routing tables' rules at the network update time
- Output:
  - All packets transmitted in the network (automatically generated by the given code)

Input Sample:
use cin

Format:
#Nodes #Dsts #Links
InsTime UpdTime SimDuration
DstID
…
LinkID Node1 Node2 oldW newW
…
Time SrcID DstID
…





e.g.,

```
5     2     8
0     100   300
1
4
0     0     1     1     3
1     0     2     1     3
2     0     3     3     1
3     1     2     1     3
4     1     3     1     1
5     2     3     1     1
6     2     4     3     1
7     3     4     1     3
10    0     1
20    1     4
110   0     1
120   1     4
```

Output Sample:
use cout

It suffices to implement recv_handler and the other components
The output will be automatically generated ☺

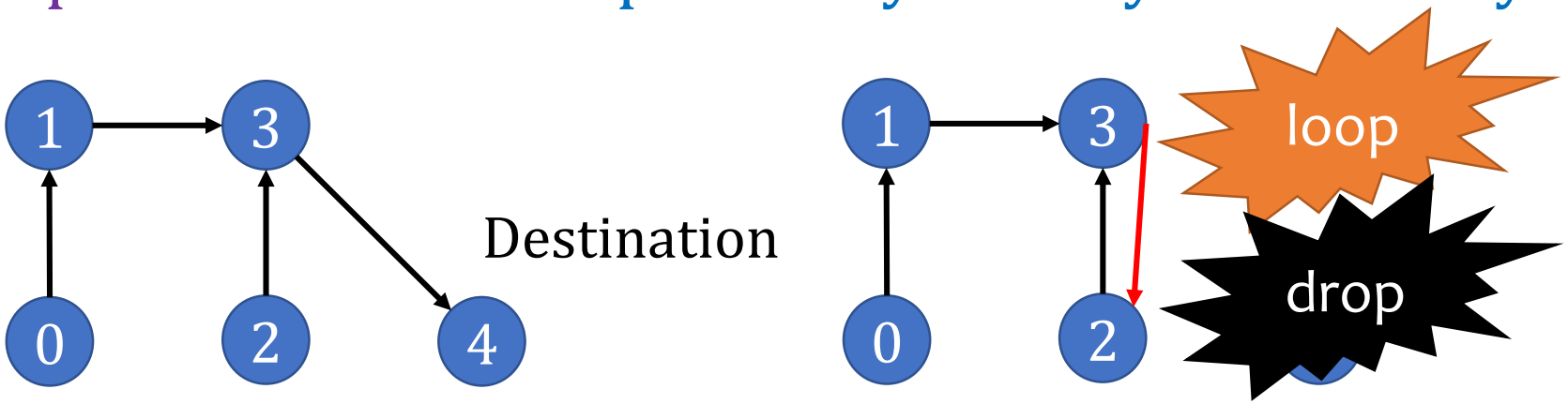
Note that the output could be different in different computers

Note

- Superb deadline: 3/29 Tue
- Deadline: 4/5 Tue
- Pass the test of our online judge platform
- Submit your code to E-course2
- Demonstrate your code remotely with TA
- **C++ Source code (only C++; compiled with g++)**
  - **Include C++ library only (i.e., no more stdio, no stdlib, …)**
- Show a good programming style

Discussion
Difficulty of Network Update in SDN

- The controller is logically-centralized
- However, the underlying mechanism is distributed
- Each switch receives the update message and updates its rule independently and asynchronously



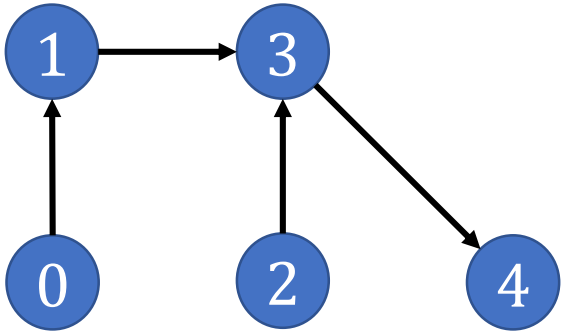
Node ID	0	1	2	3	4
To	1	3	3	4	-1

Node ID	0	1	2	3	4
To	1	3	3	2	-1

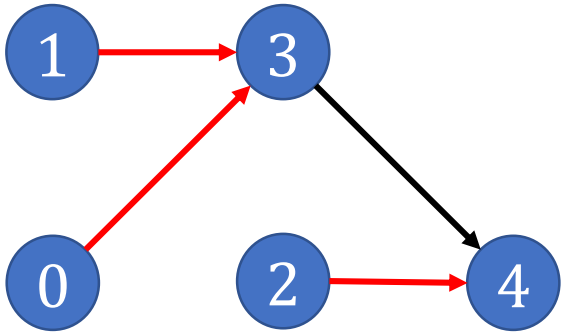
Discussion
Possible Solution to Blackholes and Loops

- Round-based update

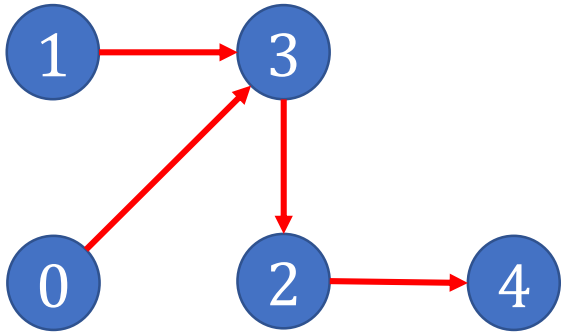
Old



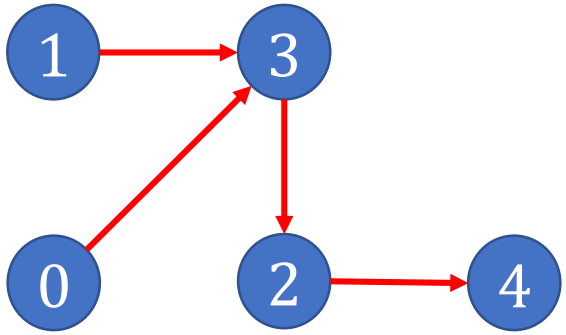
Step 1: Update 0 and 2



Step 2: Update 3



New



The Next Project Will…

- Implement more realistic links rather than simple links → class SDN_link
- Each link could have different latency
- Implement a new class SDN_controller
- Create a SDN controller and SDN switches and then connect each switch to the controller
- Implement the round-based update
- A competition may happen:
- Maybe: Less update rounds → Higher score