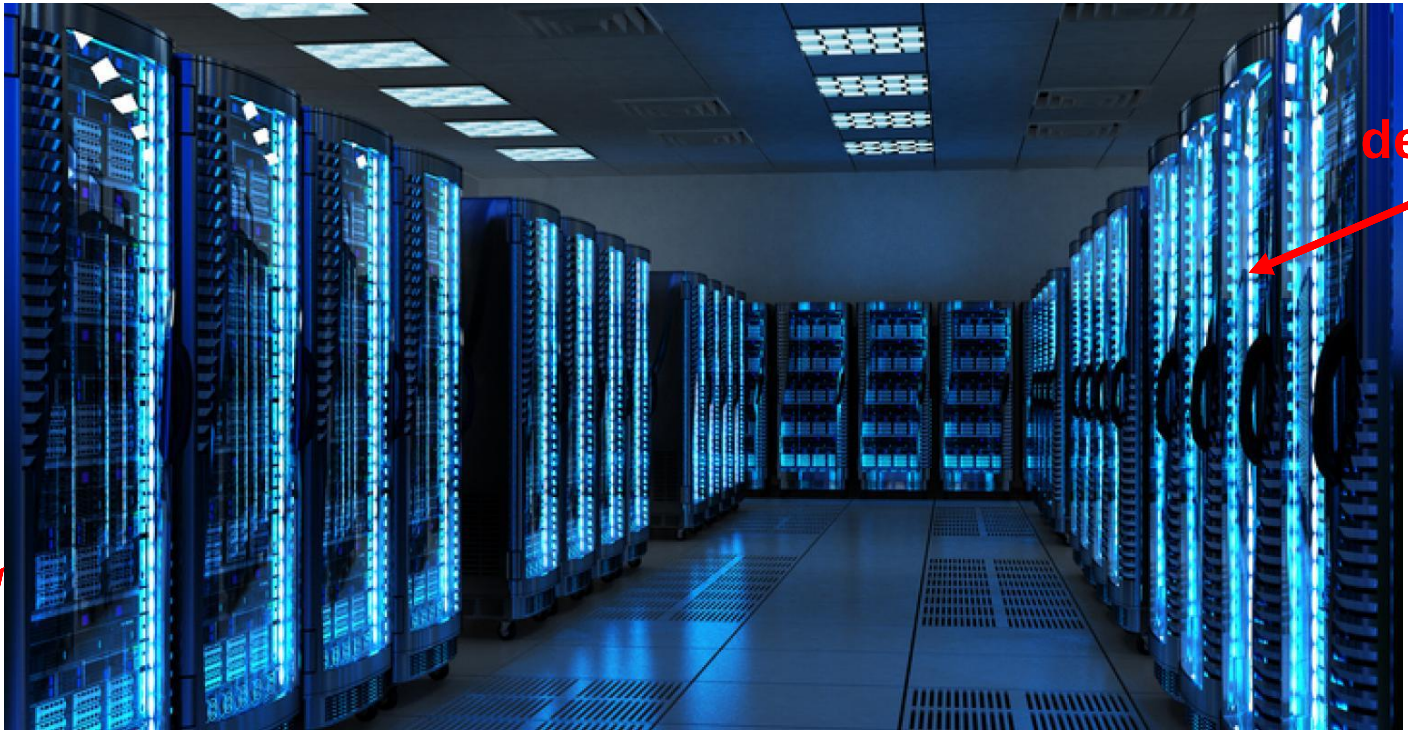
Object-Oriented Programming
Programming Project #4

Data Center

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



destination

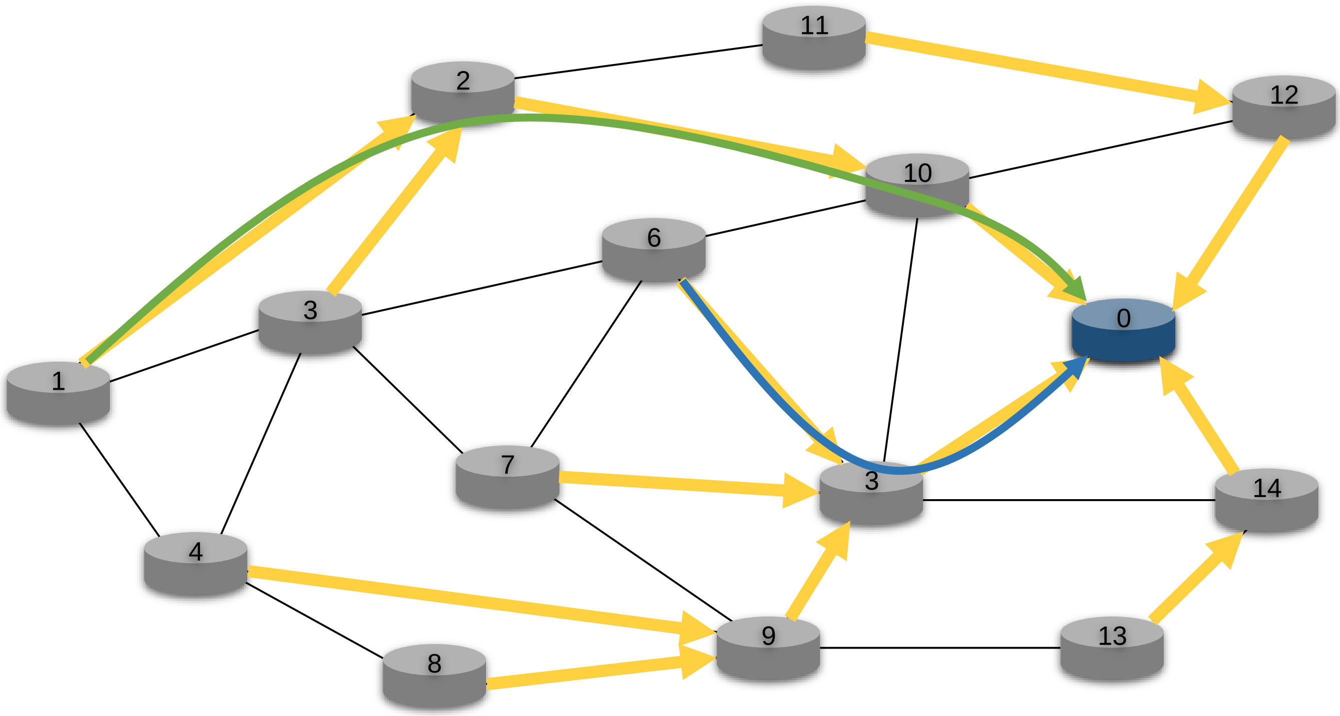
source

Switches

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

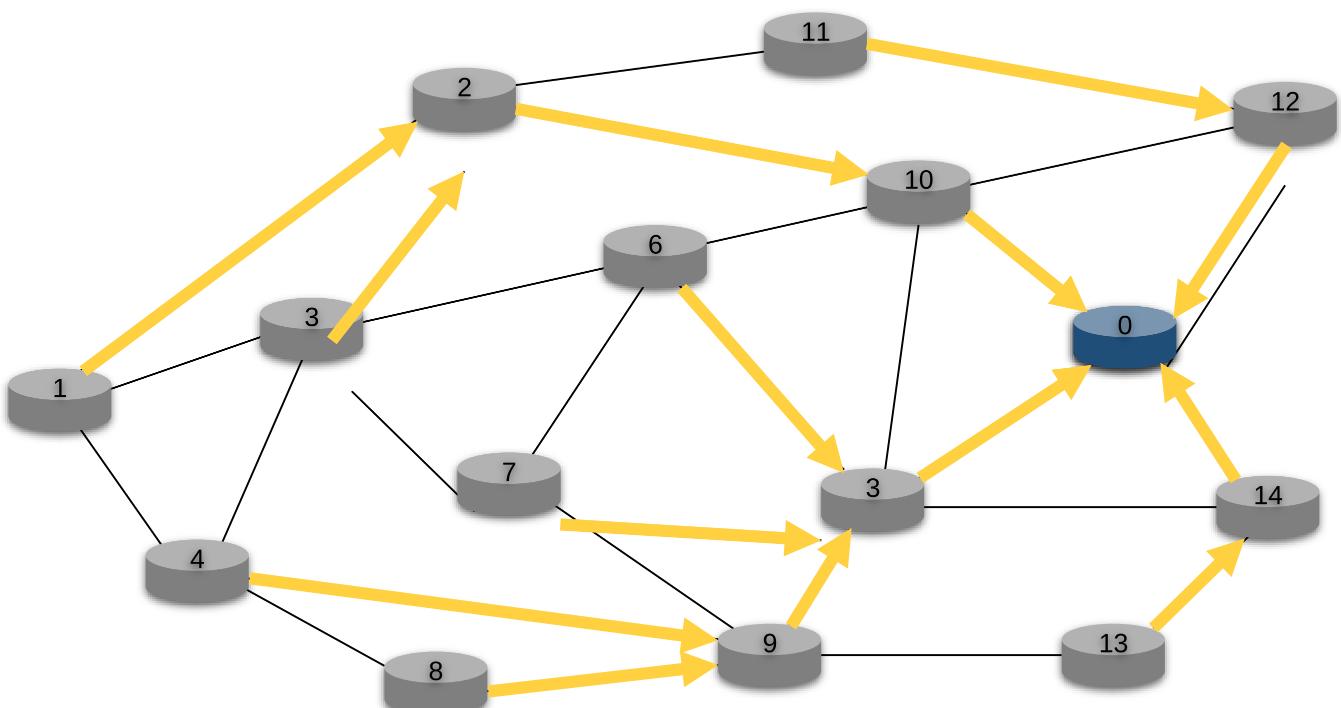
Traditional Routing Path

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



OSPF Routing Information (HW2)

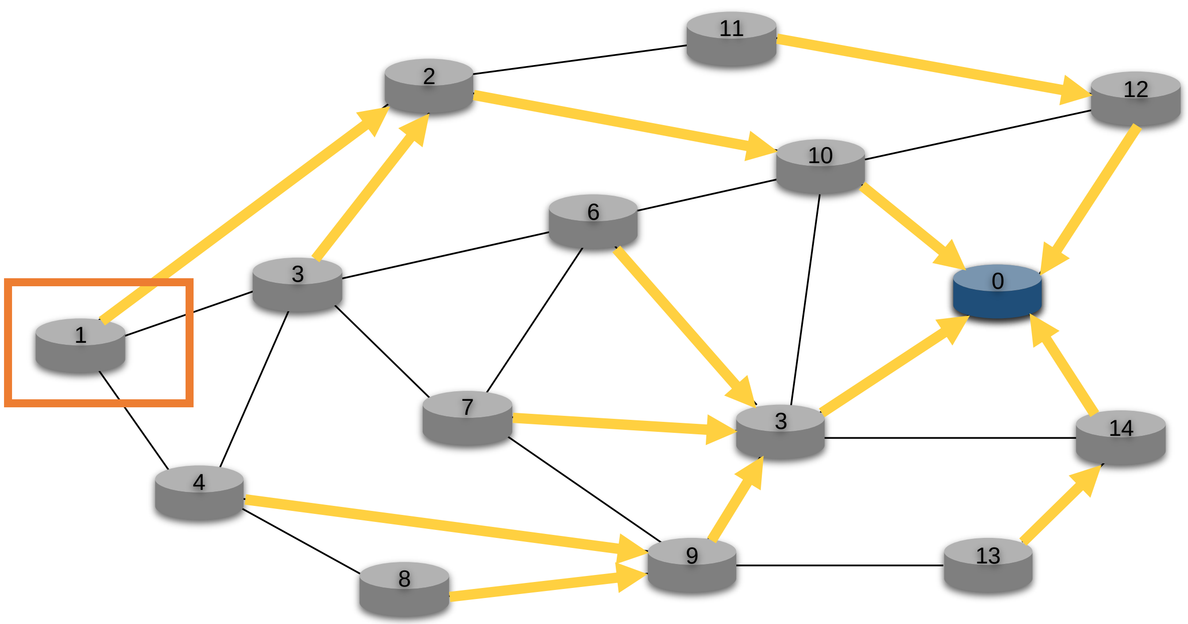
- Given: a graph with links and destinations
- Output: shortest paths towards all destinations
- Then, store the information in each node's table



OSPF Routing Table

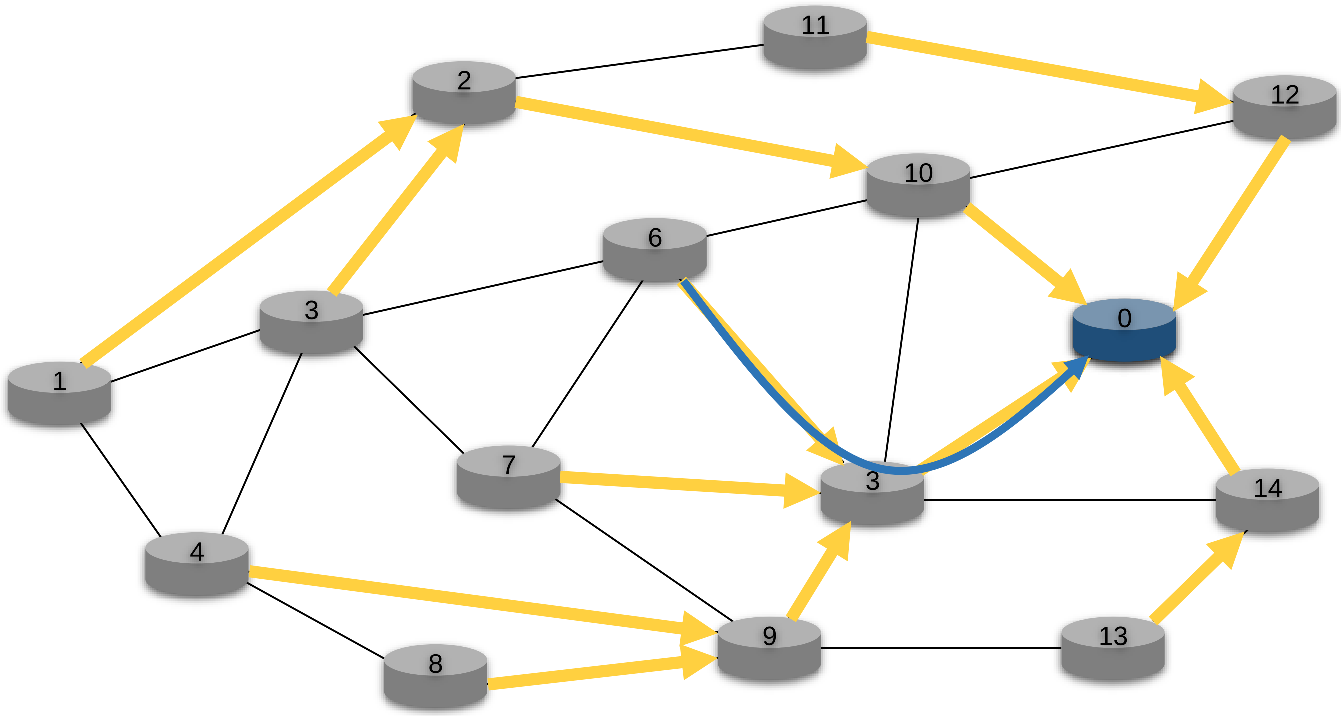
- Key: each destination
- Value: the next node (i.e., the output port)
- Node 1's table (it uses OSPF)

Destination	Next Node
0	2



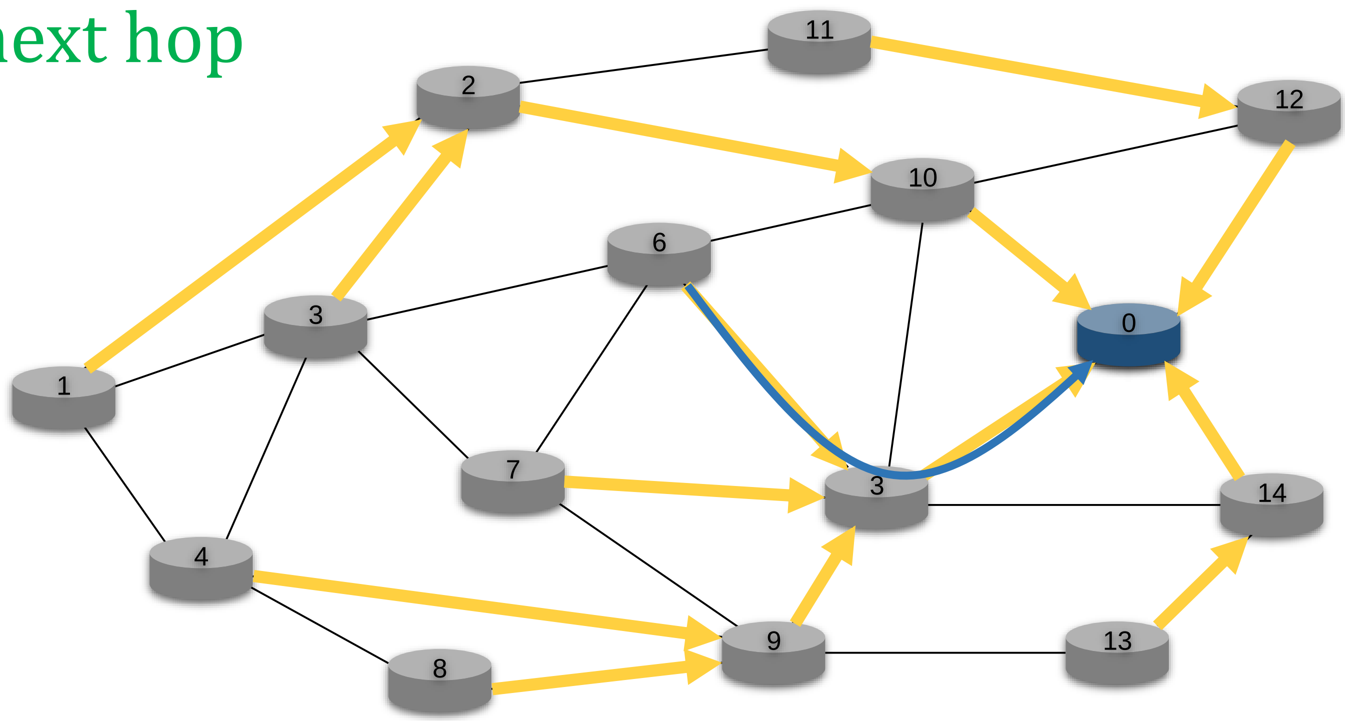
Note – Rule for Selecting the Next Hop (HW2)

- Select the node with a smaller counter (i.e., closer to the destination)
- Select the node with a smaller ID as the next hop if there is a tie (i.e., multiple candidates)



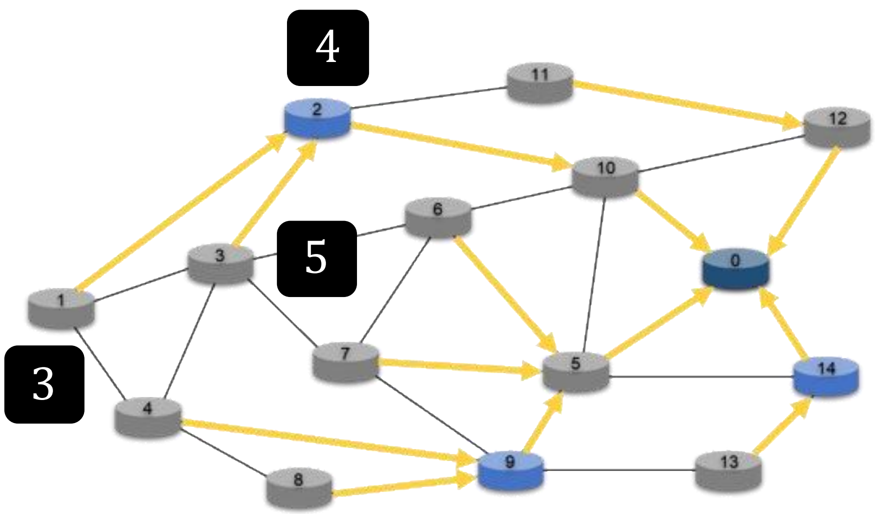
Note – Rule for Relaying TRA_ctrl_packet (HW2)

- Relay packets with a counter smaller than all my currently received counters
- Relay packets with a counter equal to my next hop's counter but with a preID smaller than my current next hop

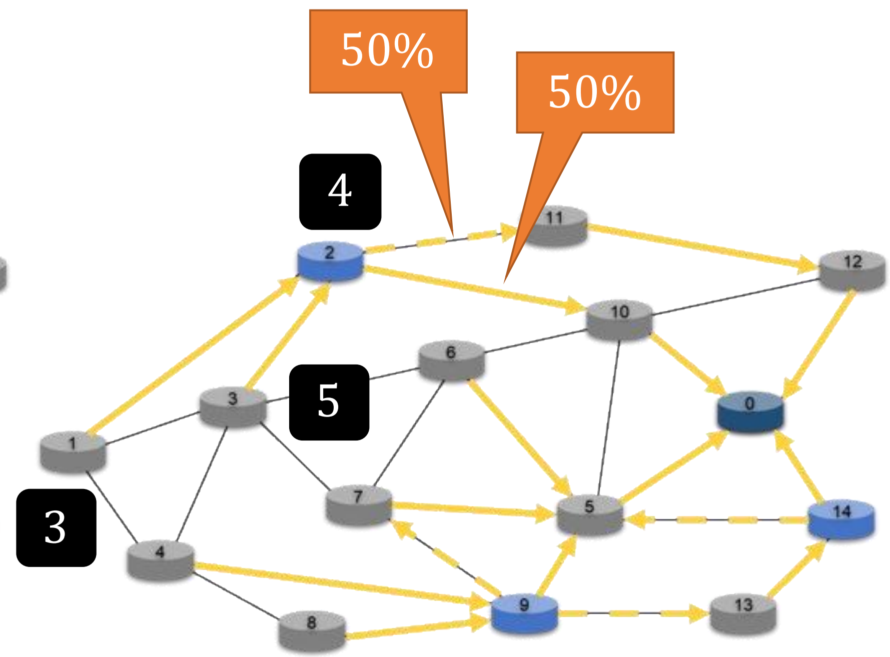


Splitting Flows with SDN switches (New)

- Given a set of SDN switches
- Goal: leverage the SDN switches to make the maximum link load smaller than that with no SDN switch



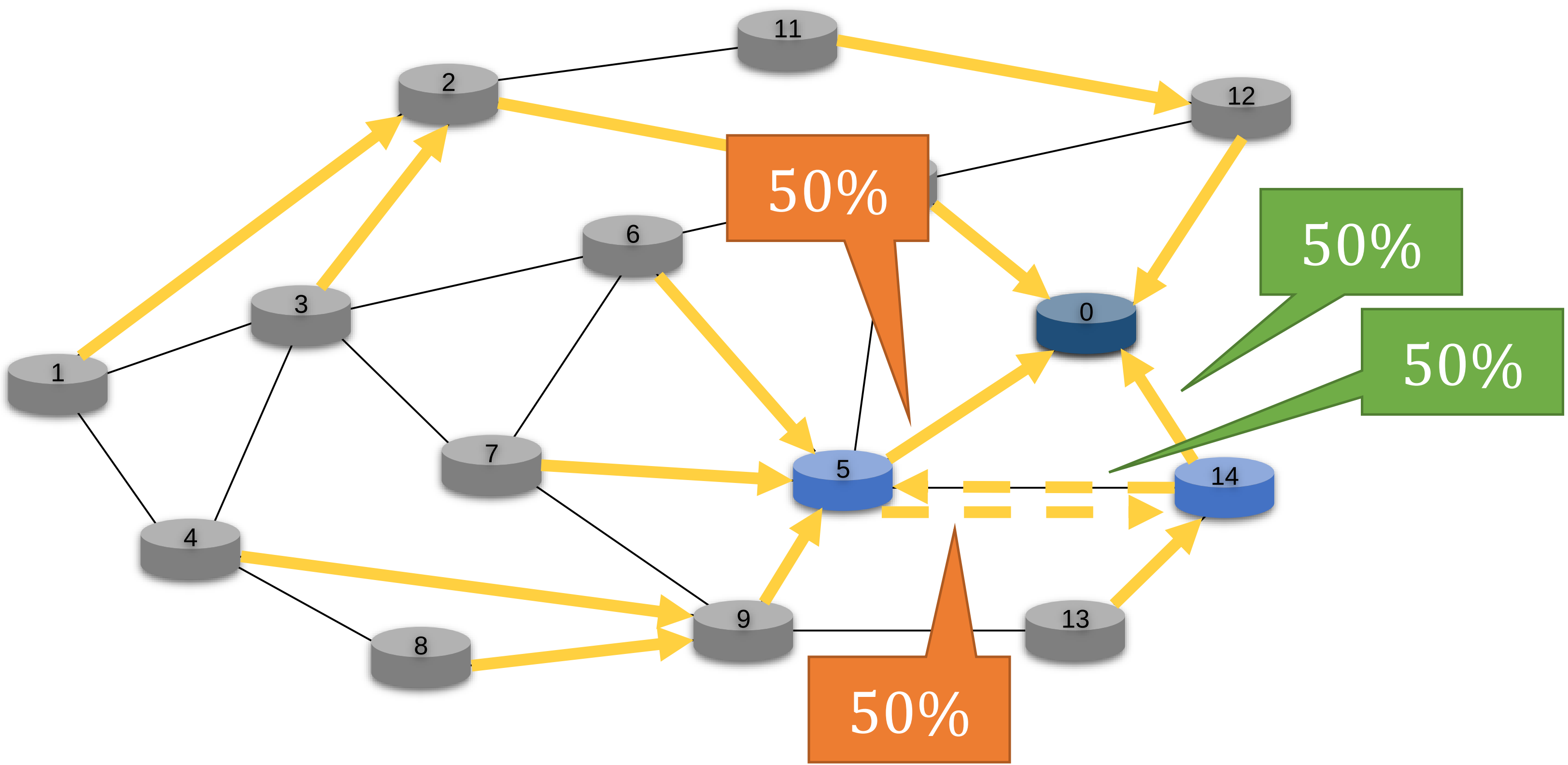
Max link load = 12



Max link load = 6

Note – No Cycle for Routing (HW1)

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



Programming Project #4: Decrease Max Link Load with SDN switches

- Input:
  - # 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 in a distributed manner
  - Invoke SDN controller to compute next hops and portions for SDN-enabled nodes (a must)
  - Route packets with different size toward destinations
- Output:
  - Each node's routing table
  - Packet exchange information will be logged automatically

Note – Create TRA Nodes and Links (HW2)

- Create traditional switches (i.e., class TRA_switch)
- Each traditional switch has an unsigned int ID
  - node::node_generator::generate(”TRA_switch", id);
- Every node only knows its neighbors
- Add the neighbors for each traditional 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., each entry in the table has <a destination ID, a next node ID>) in class TRA_switch
  - Copy and modify partial code in HW2

Note – Create SDN Nodes and Links (HW3)

- Define class SDN_switch and Create SDN_switch
  - Derived from class node (Inheritance)
- Each SDN switch has an unsigned int ID
  - node::node_generator::generate("SDN_switch",id);
- Every node only knows its neighbors
- Add the neighbors for each SDN 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, vector<pair<unsigned int, double> > > to store each entry in each switch’s table (i.e., each entry in the table has destination ID, <next nodes’ IDs, portions>) in class SDN_switch
  - Copy and modify the routing table code in HW3

Note – Define and Create SDN Controller (HW3)

- Define new class SDN_controller
  - Derived from class node (Inheritance)
- After creating the switches, create an SDN controller
  - con_id is the controller ID (after all switches' ID)
  - node_generator::generate("SDN_controller", con_id);
- Connect each SDN switch to the controller
  - node::id_to_node(switch_id)->add_phy_neighbor(con_id);
  - node::id_to_node(con_id)->add_phy_neighbor(switch_id);

Note – Generate Traditional Ctrl Packets (HW2)

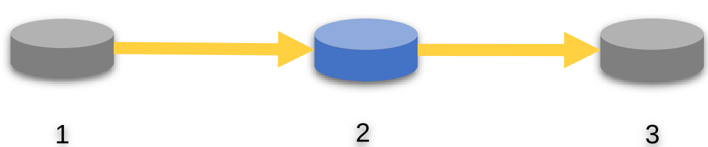
- Generate traditional ctrl packets
  - void **TRA_ctrl_packet_event**(unsigned int src, unsigned int t = event::getCurTime(), string msg = "default")
  - The function is used to initialize the distributed BFS; that is, a TRA_ctrl_packet will be generated for a source (src) with a counter 0
  - You have to implement recv_handler() in TRA_switch to forward the ctrl packet to the neighboring node again; that is, every node receiving the packet should increase the counter and broadcast the packet to its neighboring nodes to update the rule in every node's table to build the path from every node to the source
  - The source (src) will receive the TRA_ctrl_packet first (since it's src)

Note – Receive and Send Packets (1/2) (HW2/3)

- Define the rules to handle the received packet in class TRA_swtich/SDN_switch's member function recv_handler
  - void TRA_switch::recv_handler (packet *p) (or SDN_switch's)
  - **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 *p
- Check the packet type
  - if (p->type() == "TRA_data_packet")
  - if (p->type() == "TRA_ctrl_packet")
  - if (p->type() == "SDN_ctrl_packet")

Note – Receive and Send Packets (2/2) (HW2)

- Decode: Cast the packet, payload, to the right type
  - TRA_data_packet *p2 = dynamic_cast<TRA_data_packet *> (p)
  - TRA_ctrl_packet *p3 = dynamic_cast<TRA_ctrl_packet *> (p)
  - TRA_ctrl_payload *l3 = dynamic_cast<TRA_ctrl_payload *> (p3->getPayload());
  - ...
- Before sending a packet to the next hop
  - Use setPreID(id) to change the preID to the current node's ID
  - Use setNexID(id) to change the nexID to the next hop node's ID
  - Please check all the columns in the header



PreID = 1
NexID = 2

PreID = 2
NexID = 3

Note – Generate SDN Ctrl Packets (HW3)

- Generate SDN ctrl packets
  - void **SDN_ctrl_packet_event**(unsigned int con_id, unsigned int id, unsigned int mat, unsigned int act, double per, unsigned int t = event::getCurTime(), string msg = "default")
  - A packet will be generated for the controller and sent to the node (id) at time t (optional) to update a specific rule in the node's table
  - mat: the destination in the node's routing table
    act: the next hop in the node's routing table
    per: the portion of flow sent to the next node
  - The node can use getMatID(), getActID(), and getPer() of SDN_ctrl_payload to get mat, act, and per from the SDN_ctrl_packet
  - The controller will receive the SDN_ctrl_packet first (since the controller is src)

Note – Invoke Computing Portions (New)

- Invoke SDN controller to compute portions
  - void **SDN_invoke_packet_event**(unsigned int con_id, vector<vector<double>> traffic_matrix, unsigned int t = event::getCurTime(), string msg = "default")
  - A packet will be generated for the controller at time t (optional) to notify the controller to compute the portions for each SDN switch based on the traffic matrix (traffic_matrix)
  - The controller will receive the SDN_invoke_packet first (since the controller is src and also dst)
  - The controller can know that the SDN_invoke_packet is used to notify itself to compute portions
  - Move the code for computing routing tables in SDN switches into the recv_handler function of the SDN controller
  - The function recv_handler in SDN controller is allowed to use **node::id_to_node(id)**

Note – Generate Data and Ctrl Packets (New)

- Generate data packets
  - void **data_packet_event**(unsigned int src, unsigned int dst, double size, unsigned int t = 0, string msg="default")
  - A TRA_data_packet with the flow size (size) will be generated for a source (src) and sent to a destination (dst) at time t
  - The source (src) will receive the TRA_data_packet first (since it's src)

Inheritance

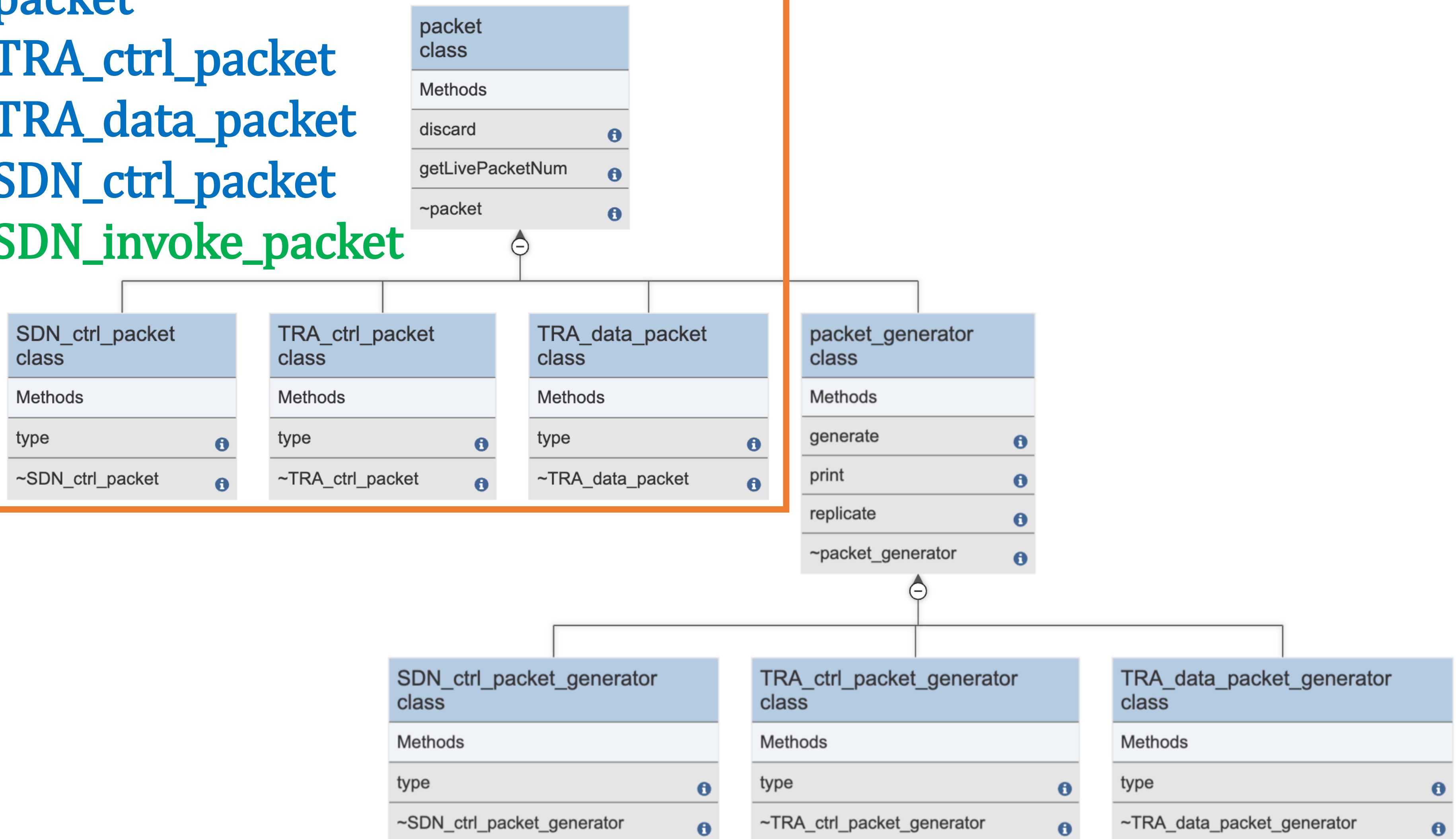
packet

TRA_ctrl_packet

TRA_data_packet

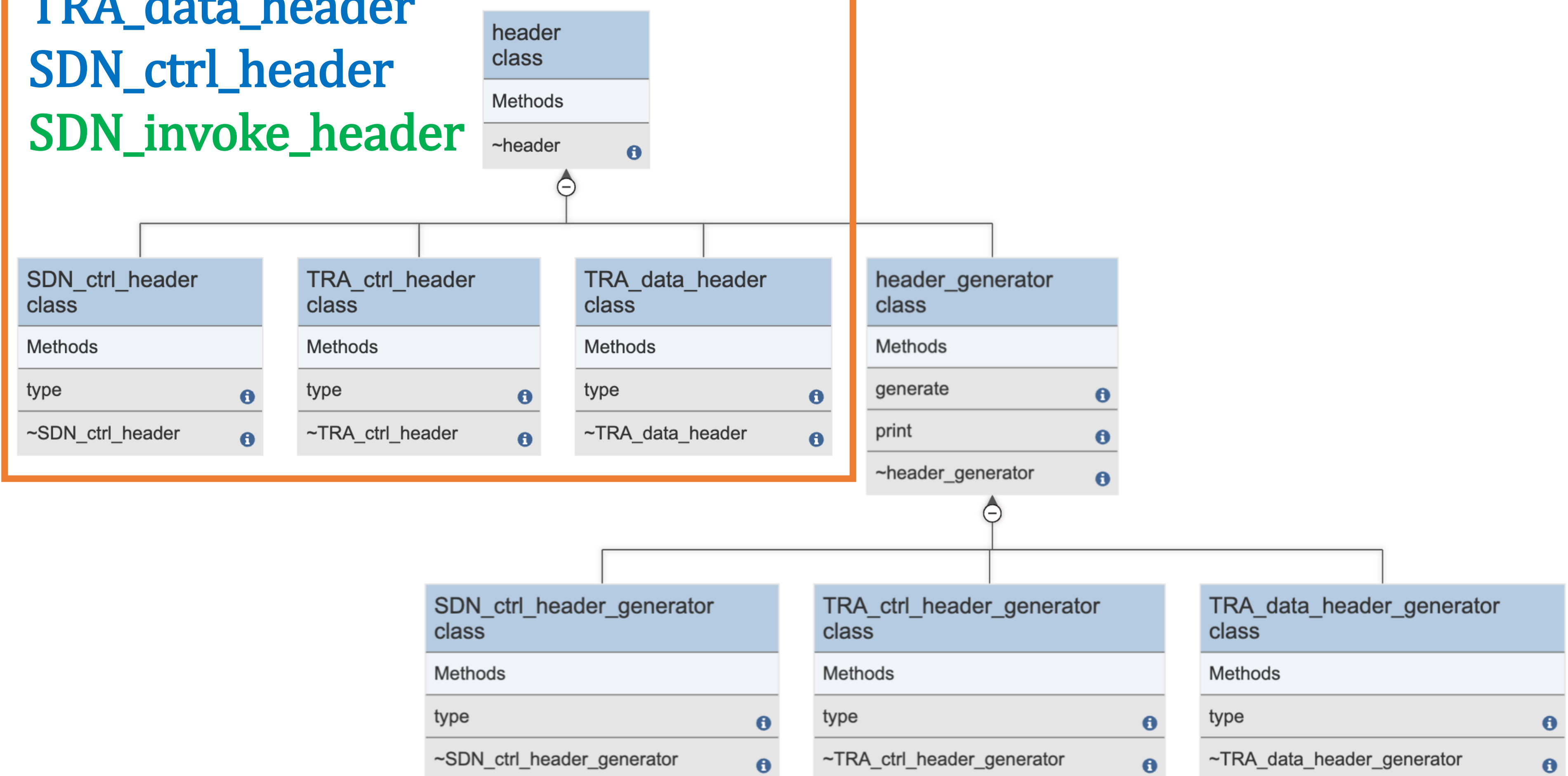
SDN_ctrl_packet

SDN_invoke_packet



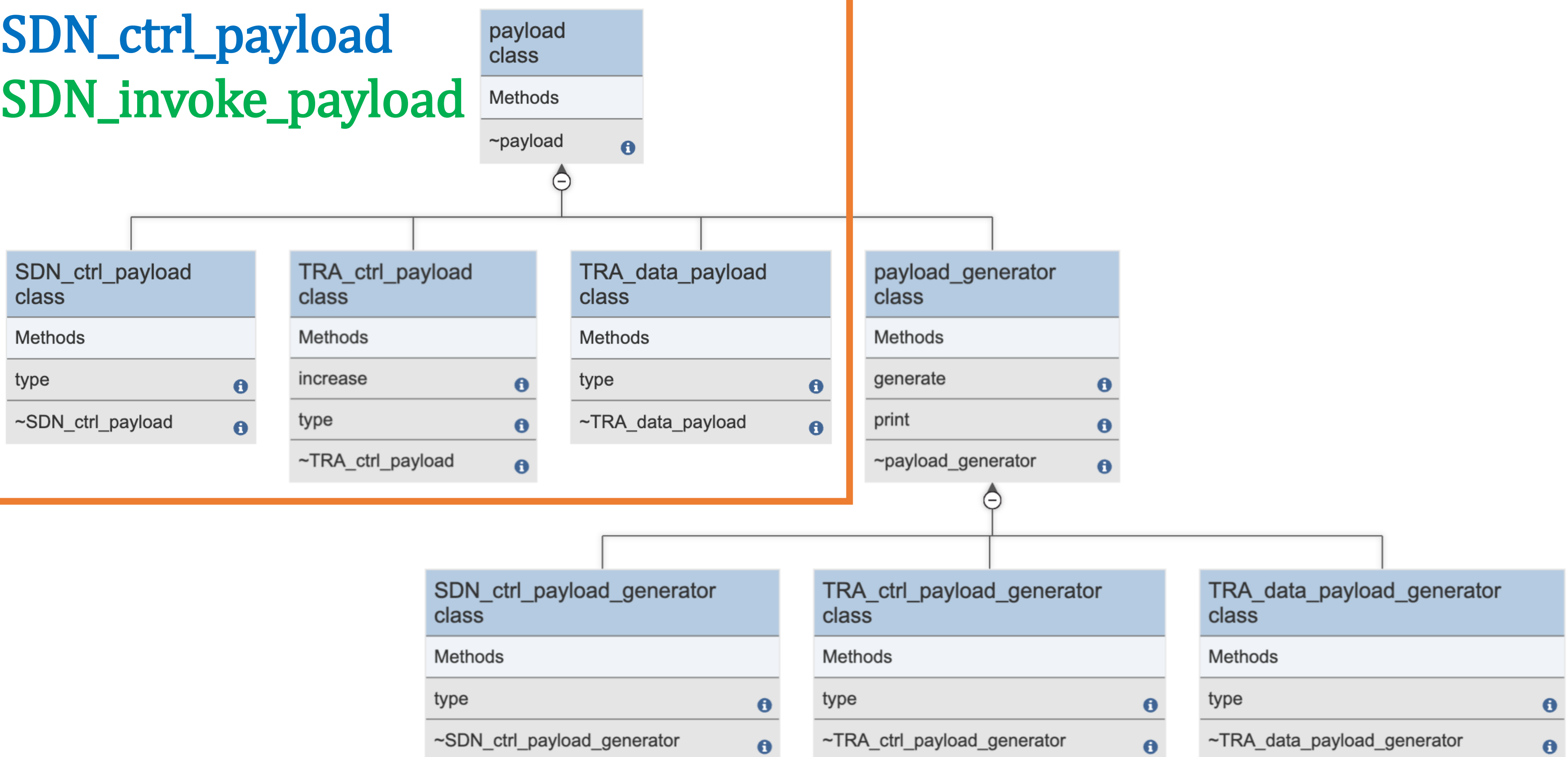
Inheritance

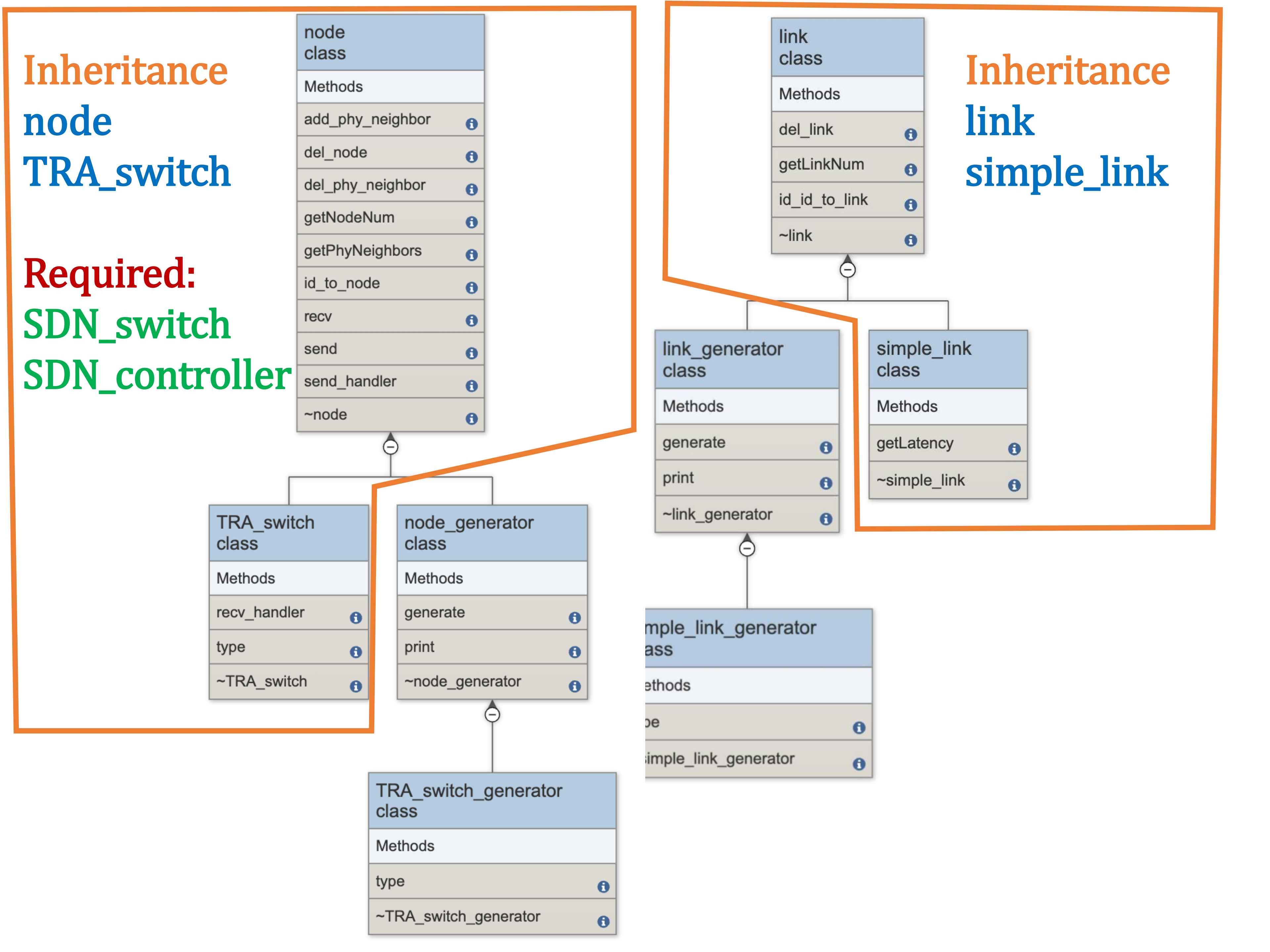
header
TRA_ctrl_header
TRA_data_header
SDN_ctrl_header
SDN_invoke_header



Inheritance

payload
TRA_ctrl_payload
TRA_data_payload
SDN_ctrl_payload
SDN_invoke_payload





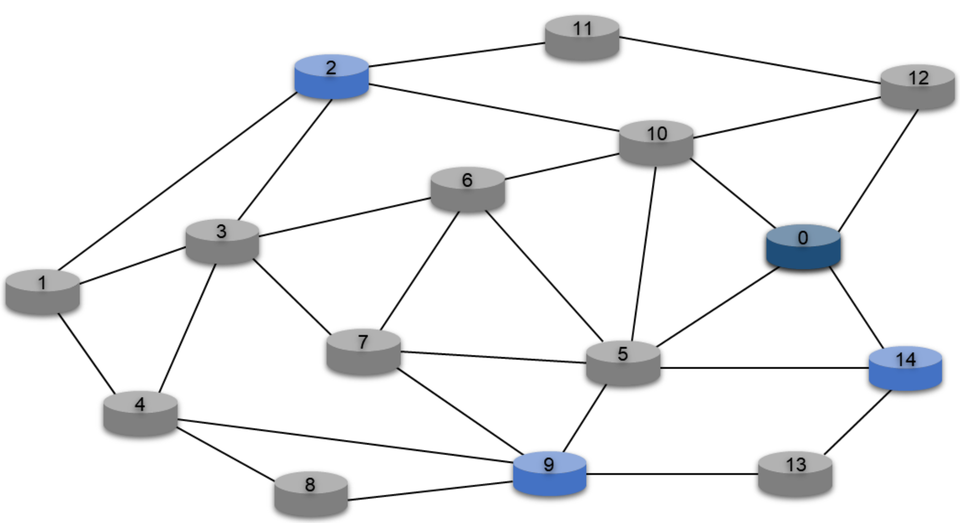
Inheritance
node
TRA_switch
Required:
SDN_switch
SDN_controller
node
class
Methods
add_phy_neighbor
del_node
del_phy_neighbor
getNodeNum
getPhyNeighbors
id_to_node
recv
send
send_handler
~node
TRA_switch
class
Methods
recv_handler
type
~TRA_switch
node_generator
class
Methods
generate
print
~node_generator
TRA_switch_generator
class
Methods
type
~TRA_switch_generator
link
class
Methods
del_link
getLinkNum
id_id_to_link
~link
Inheritance
link
simple_link
link_generator
class
Methods
generate
print
~link_generator
simple_link
class
Methods
getLatency
~simple_link
mple_link_generator
ass
ethods
pe
imple_link_generator

Input Sample:
use cin

Format:

#Nodes #SDN_Nodes #Dsts #Links #Pairs SimTime InvokeTime
SDN_NodeID_List
DstID_List
NodeID BroadcastTime
...
LinkID Node1 Node2
...
FlowID Src Dst FlowSize StartTime
...

```
15 3 1 28 3 300 150
2 9 14
0
0 20
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 3 200
1 2 0 4 220
2 3 0 5 250
```



Output Sample:
use cout

You have to print the routing table for each node after event::start_simulate();

The way to print the routing table is the same as that in HW1/HW2/HW3

The remaining output will be automatically generated ☺

Note that the output could be different in different computers

Output Sample (continue):
use cout

The example may not be optimal

Format:

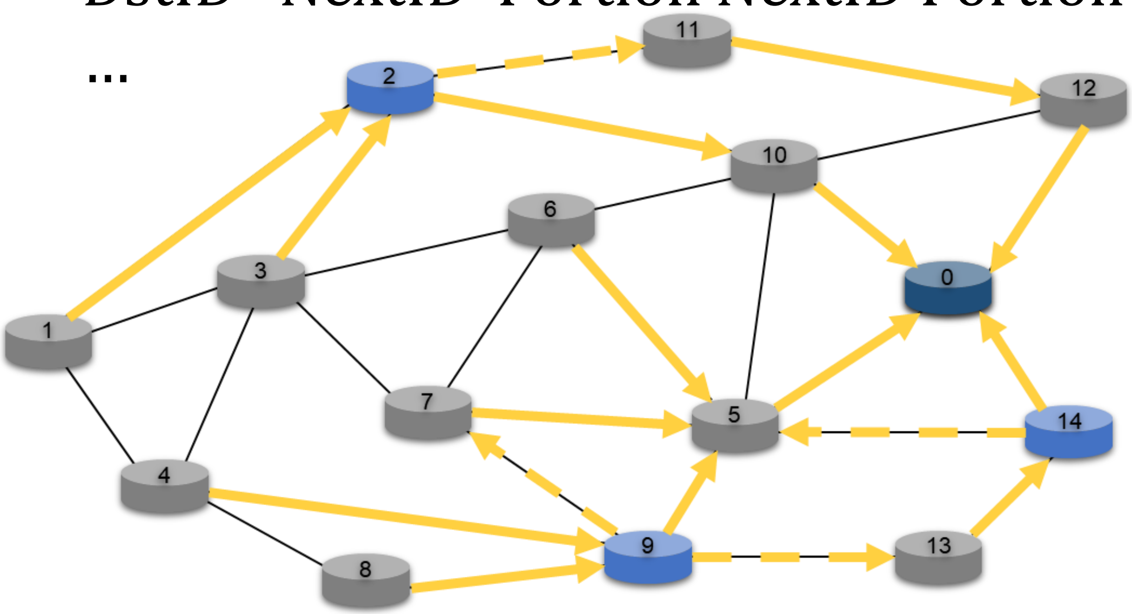
The automatic printing (you can't change)

UpgradedNodeIDList

NodeID

DstID NextID Portion NextID Portion…

…



e.g.,

2 9 14
0
0 0
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%

Its own ID

Note

- Superb deadline: 6/18 Sun (you have more than 2 weeks)
- Deadline: 6/21 Wed (you have about 3 weeks)
- Pass the test of our online judge platform
- Submit your code to E-course2
- Demonstrate your code remotely with TA (before 6/18 or 6/21)
- **C++ Source code (only C++; compiled with g++)**
  - **Please use C++ library (i.e., no stdio, no stdlib)**
  - **Please use new and delete instead of malloc and free**
- Show a good programming style