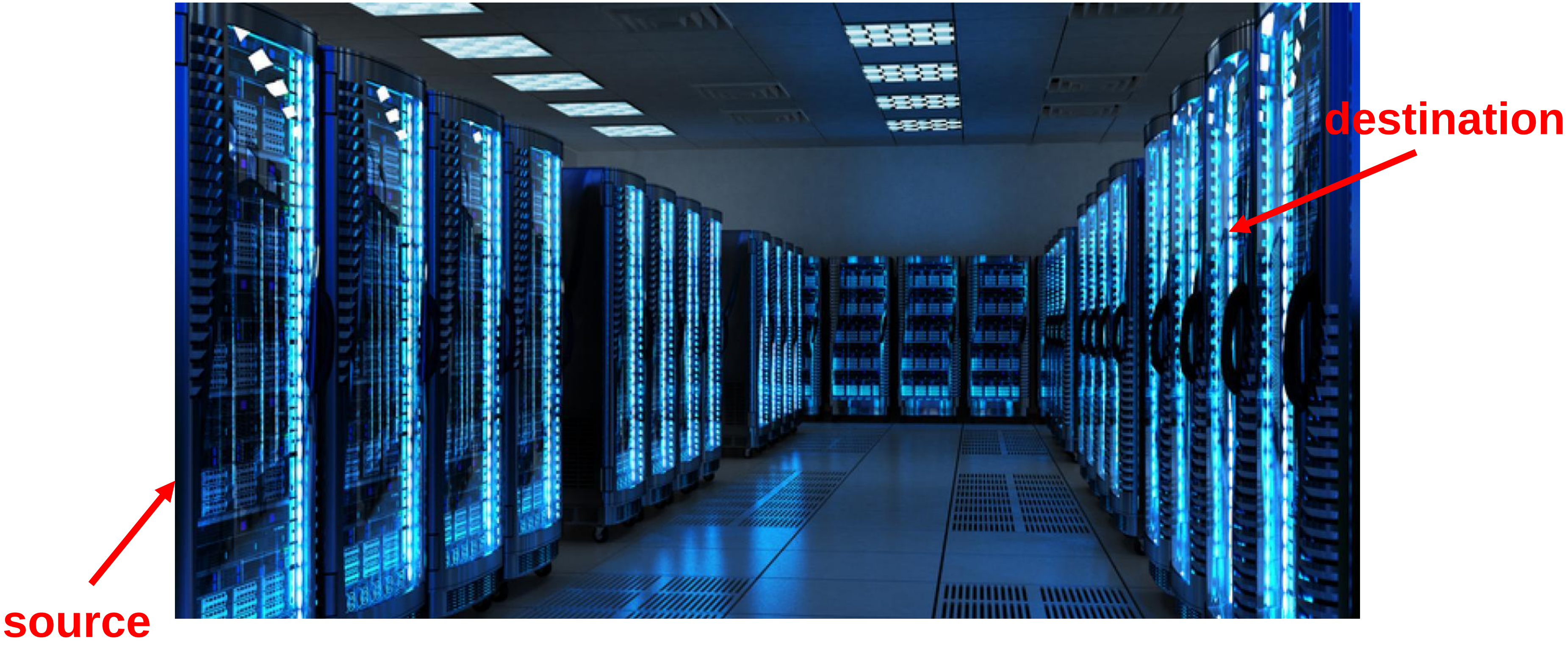
Object-Oriented Programming
Programming Project #2+3

Data Center

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

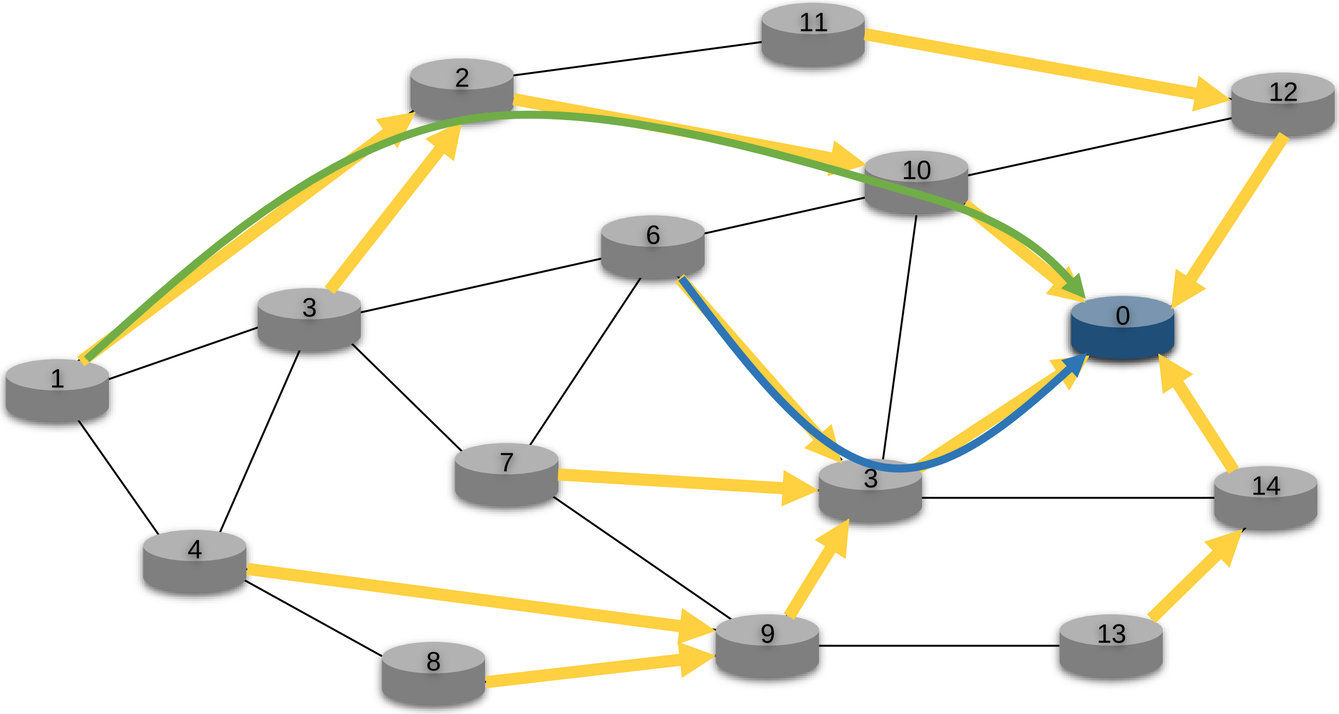


Switches

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

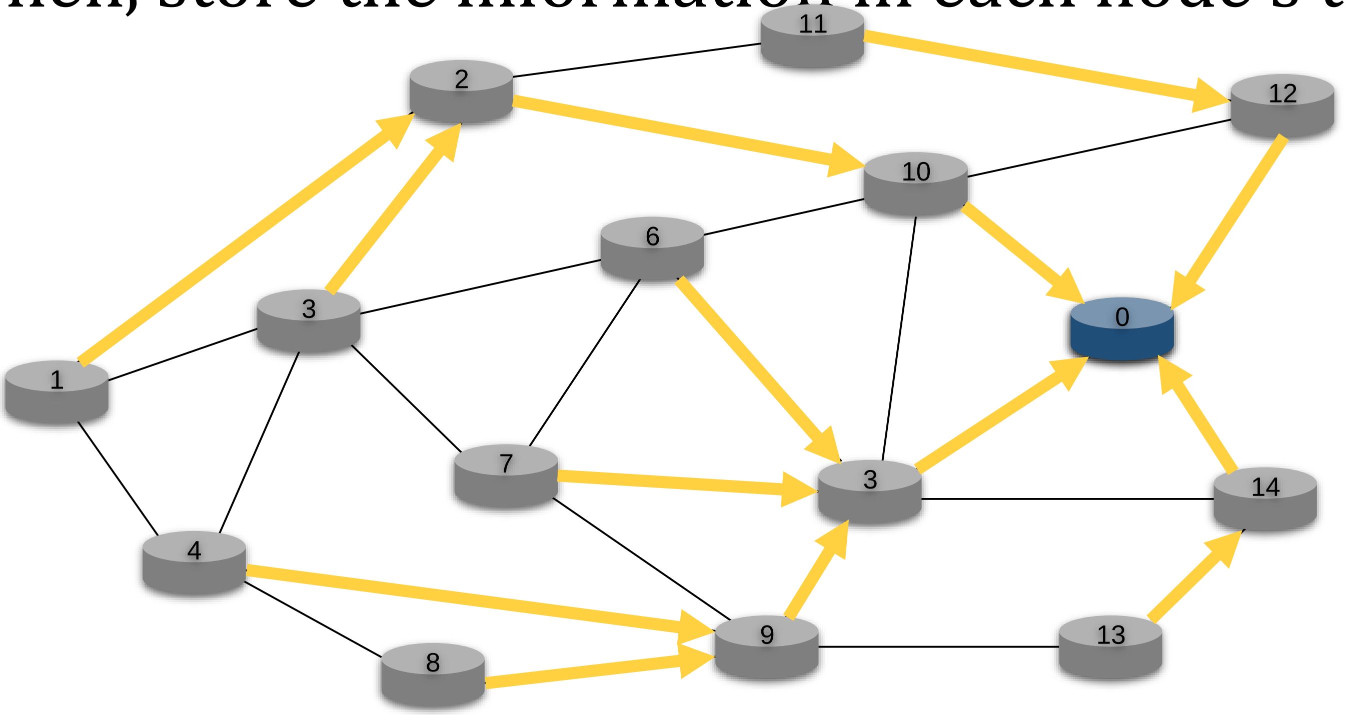
Traditional Routing Path

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



OSPF Routing Information

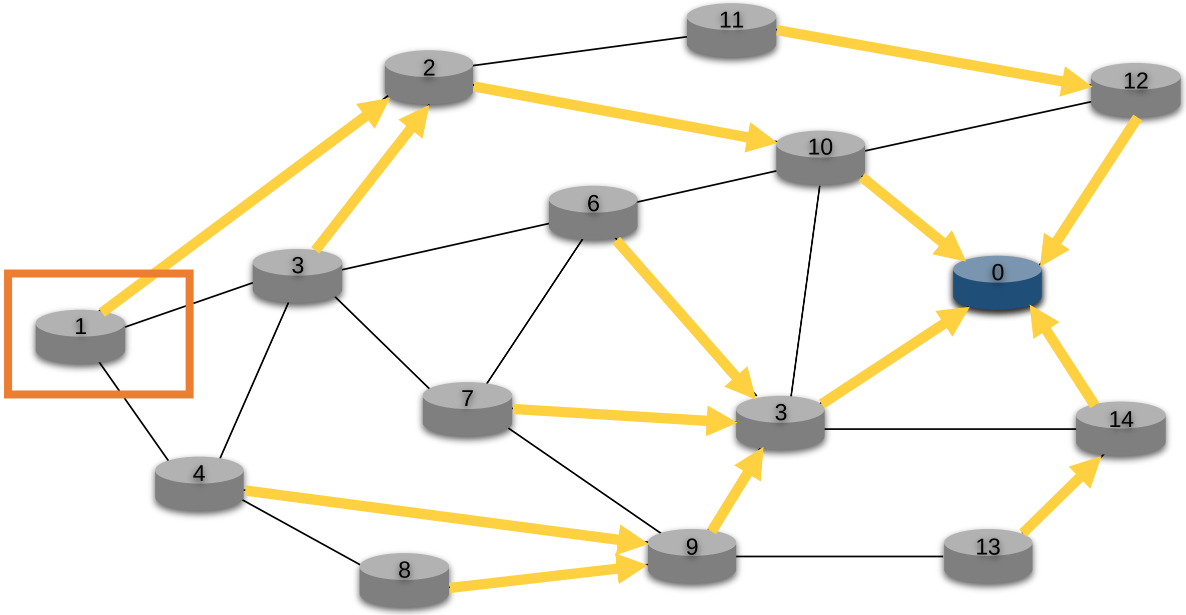
- Given: a graph with links and destinations
- Output: shortest paths towards all destinations
- Then, store the information in each node's table



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

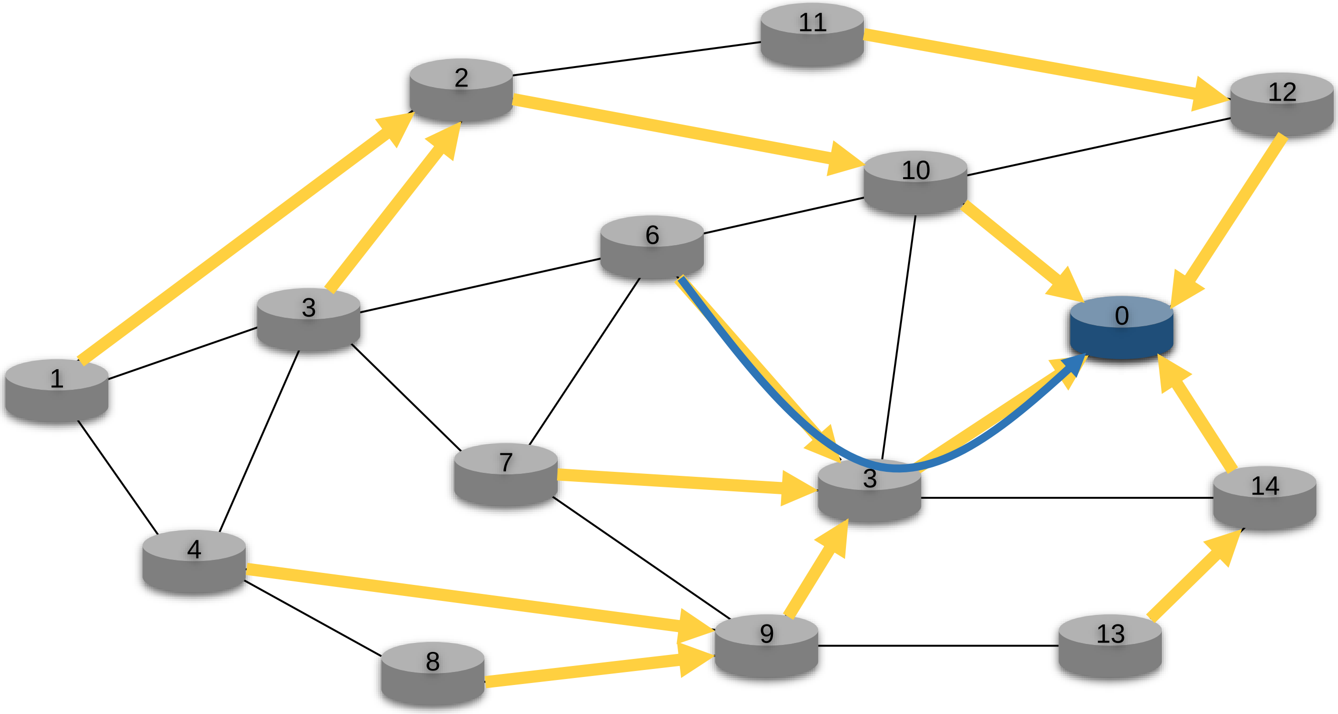


Programming Project #2:
Implement OSPF in Traditional Networks

- Input:
  - # nodes, # destinations, # links, and #pairs
  - Destinations (ID)
  - Links between nodes
  - Traffic matrix (flow size for each pair)
- Procedure:
  - Compute shortest paths to each destination
    using distributed BFS with a counter
    (choose the node with a smaller ID if there is a tie)
  - Forward packets with the help of routing tables
- Output:
  - Every node's routing table
  - Packet exchange information will be logged automatically

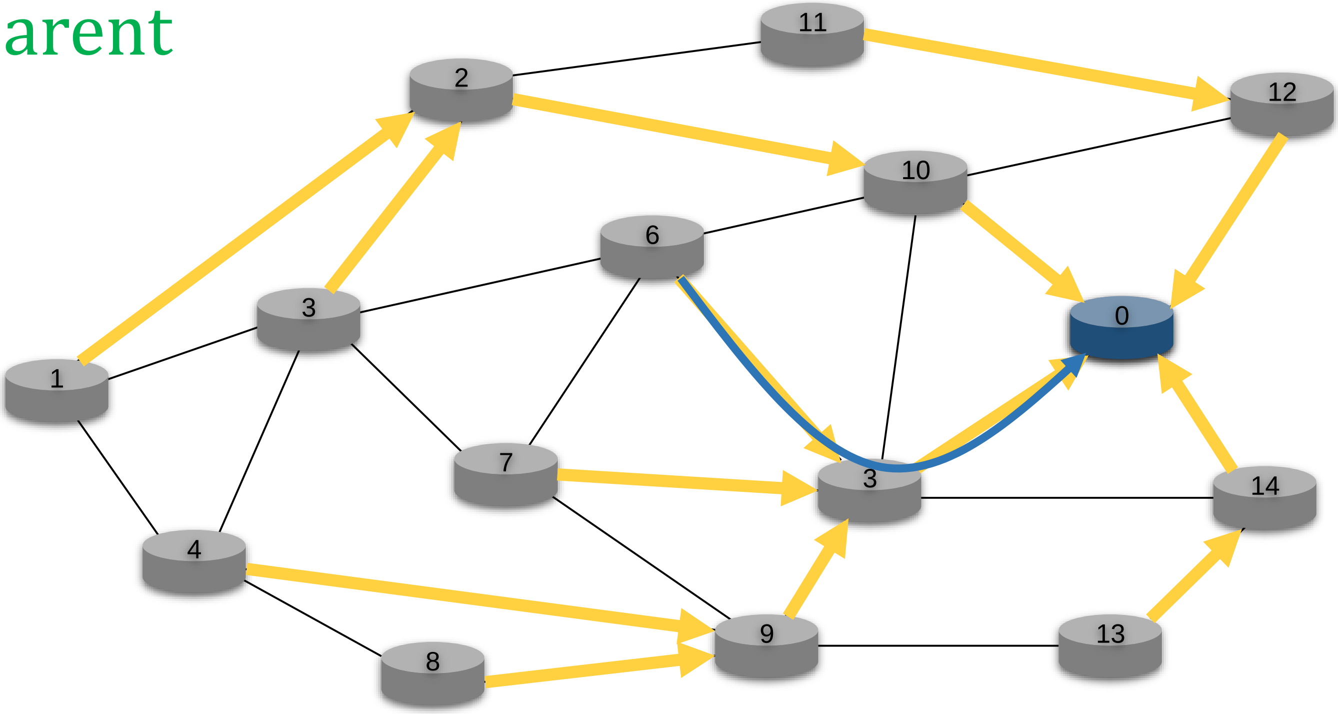
Note – Rule for Selecting the Next Hop

- 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

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



Note – Create TRA Nodes and Links

- Create traditional switches (i.e., class TRA_switch)
- Each switch has an unsigned int ID
    - node::node_generator::generate("TRA_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., each entry in the table has <a destination ID, a next node ID>) in class TRA_switch
    - Copy and modify partial code in HW1

Note – Generate Data and Ctrl Packets

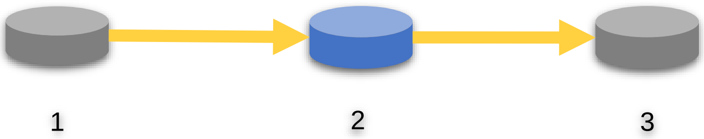
- Generate data packets
  - void **data_packet_event**(unsigned int src, unsigned int dst, unsigned int t = 0, string msg="default")
  - A TRA_data_packet will be generated for a source (src) and sent to a destination (dst) at time t
  - The source (src) will receive the TRA_data_packet first (since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

- Define the rules to handle the received packet in class TRA_swtich's member function recv_handler
  - void TRA_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() == "TRA_data_packet")
  - if (p->type() == "TRA_ctrl_packet")

Note – Receive and Send Packets (2/2)

- 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());
  - ...
  - Will be explained in the later chapters
- 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

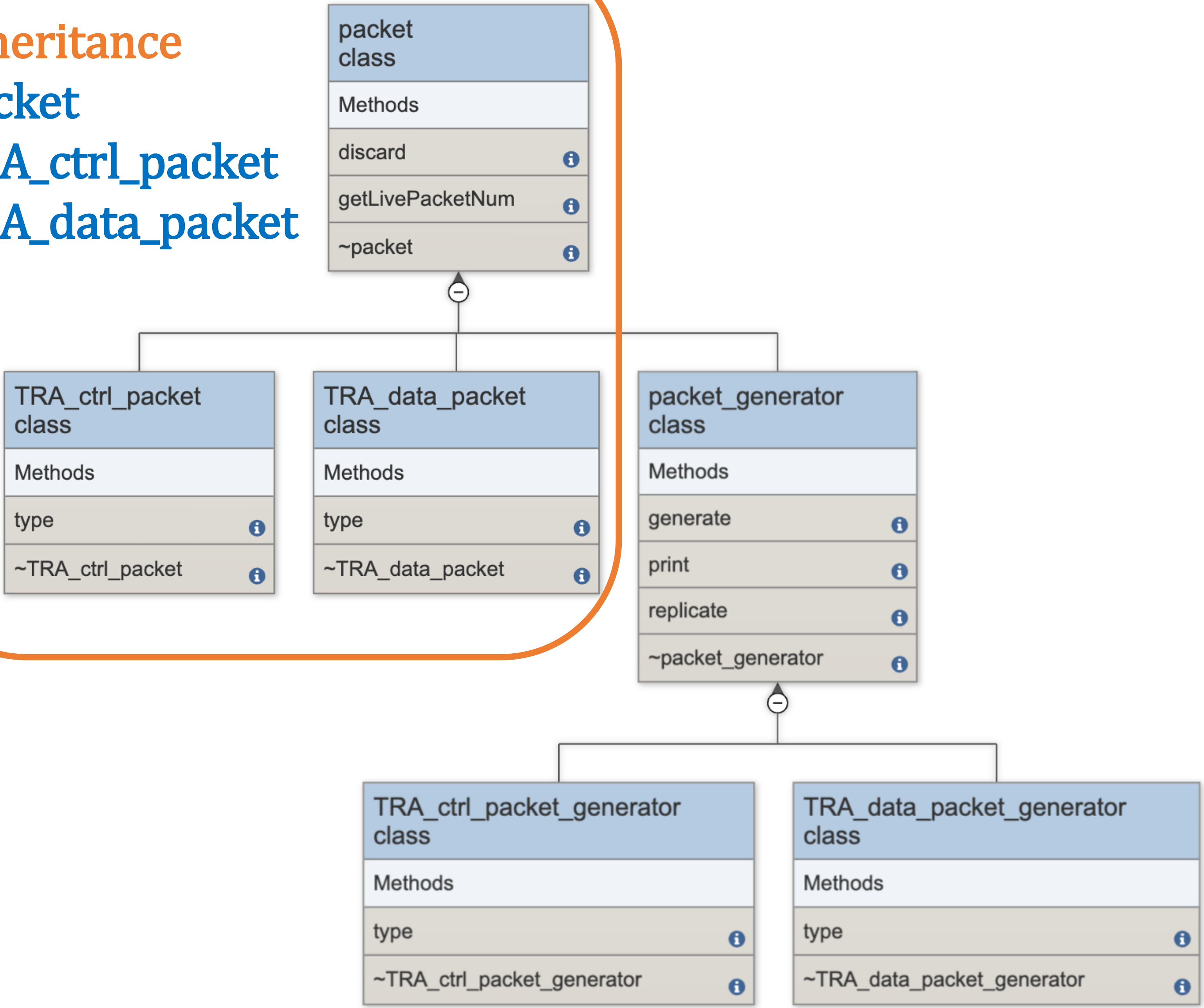
PreID = 2
NexID = 3

Inheritance

packet

TRA_ctrl_packet

TRA_data_packet



packet
class

Methods

discard

getLivePacketNum

~packet

TRA_ctrl_packet
class

Methods

type

~TRA_ctrl_packet

TRA_data_packet
class

Methods

type

~TRA_data_packet

packet_generator
class

Methods

generate

print

replicate

~packet_generator

TRA_ctrl_packet_generator
class

Methods

type

~TRA_ctrl_packet_generator

TRA_data_packet_generator
class

Methods

type

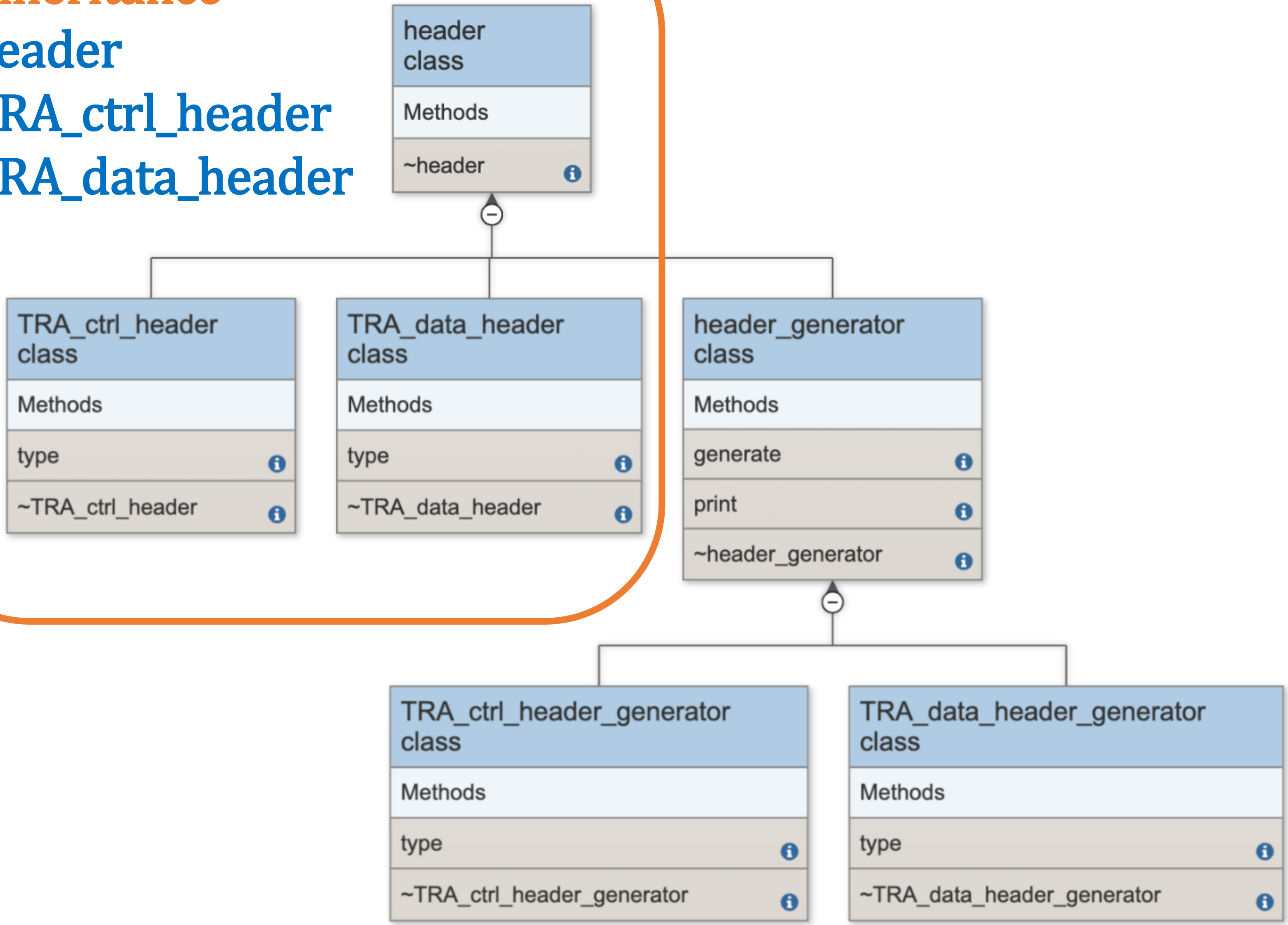
~TRA_data_packet_generator

Inheritance

header

TRA_ctrl_header

TRA_data_header

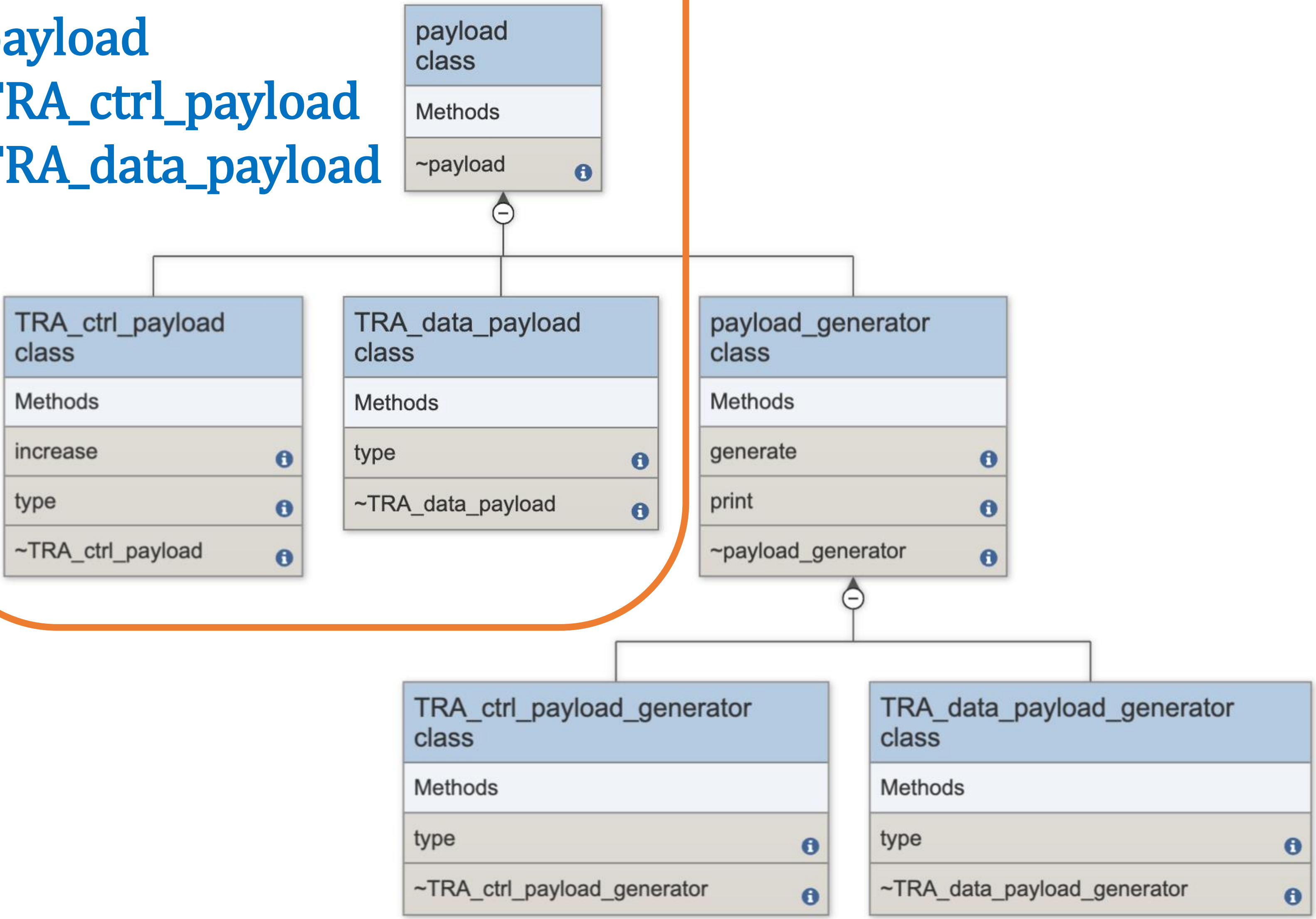


Inheritance

payload

TRA_ctrl_payload

TRA_data_payload



payload class

Methods

~payload

TRA_ctrl_payload class

Methods

increase

type

~TRA_ctrl_payload

TRA_data_payload class

Methods

type

~TRA_data_payload

payload_generator class

Methods

generate

print

~payload_generator

TRA_ctrl_payload_generator class

Methods

type

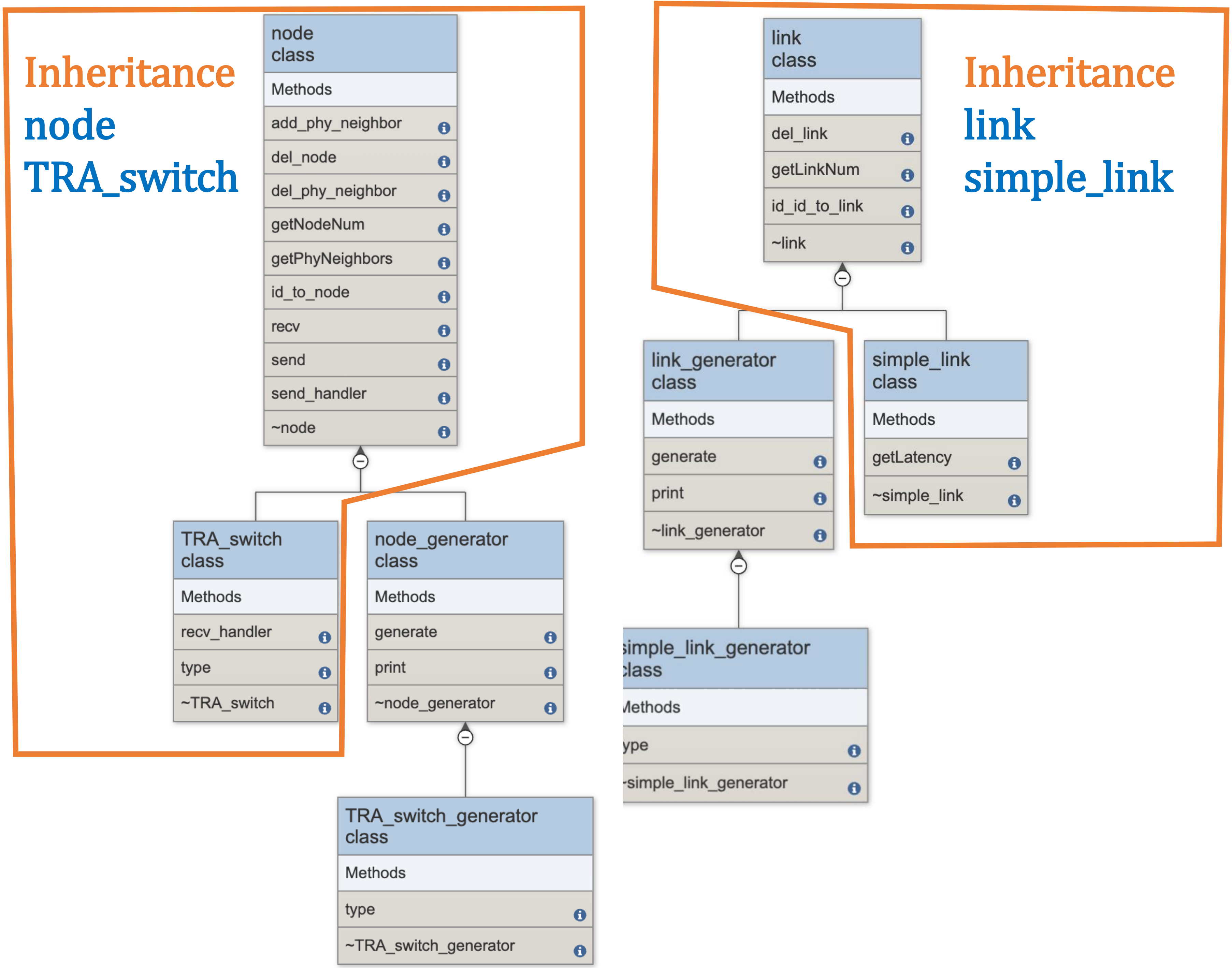
~TRA_ctrl_payload_generator

TRA_data_payload_generator class

Methods

type

~TRA_data_payload_generator



Inheritance
node
TRA_switch

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

Inheritance
link
simple_link

link class
Methods
- del_link
- getLinkNum
- id_id_to_link
- ~link

link_generator class
Methods
- generate
- print
- ~link_generator

simple_link class
Methods
- getLatency
- ~simple_link

simple_link_generator class
Methods
- type
- ~simple_link_generator

Input Sample:
use cin

Format:

#Nodes #Dsts #Links #Pairs SimTime

DstID_List

NodeID BroadcastTime

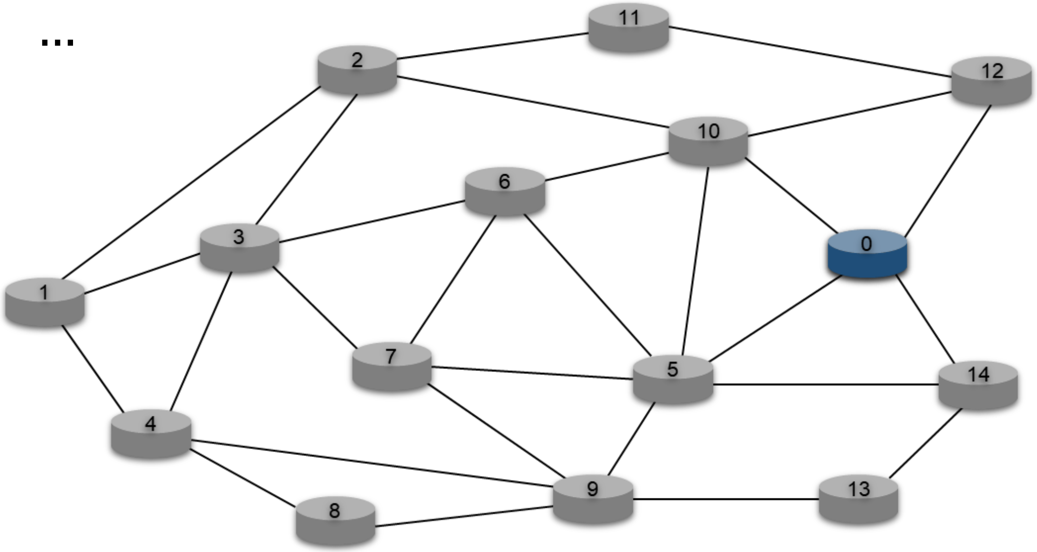
...

LinkID Node1 Node2

...

FlowID Src Dst FlowSize StartTime

...



```
15  1  28  3  300
0
0  100
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  150
1  2  0  4  200
2  3  0  5  120
```

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

The remaining output will be automatically generated ☺

Note that the output could be different in different computers

Output Sample (continue): use cout

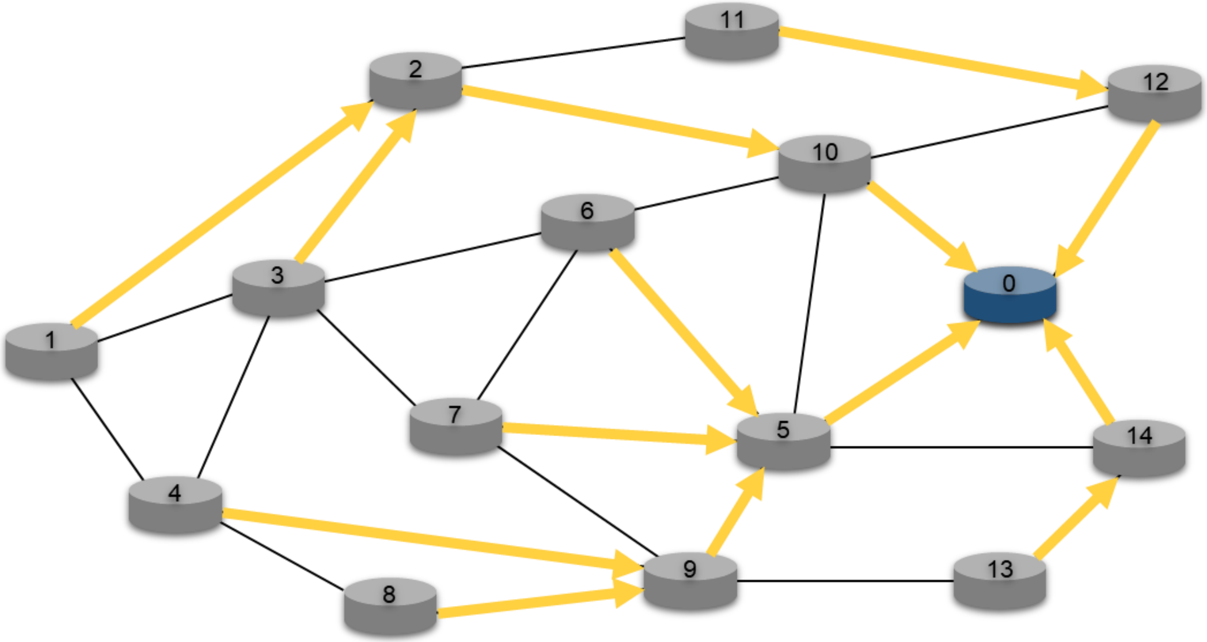
Format:

The automatic printing (you can't change)

NodeID

DstID   NextID

...

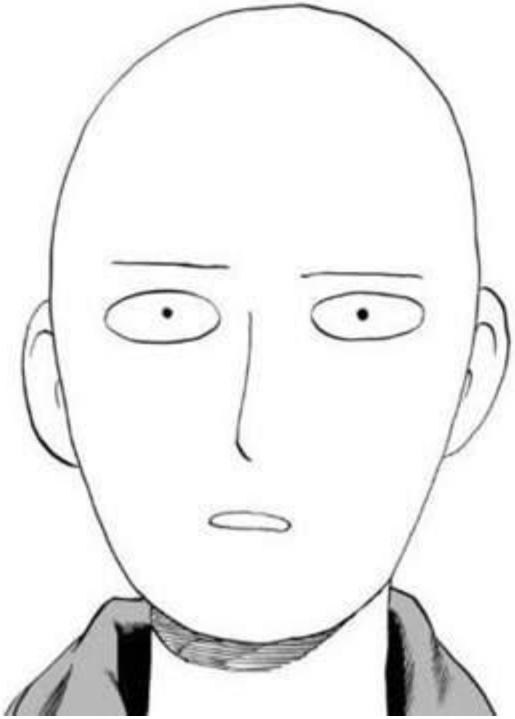


e.g.,

Its own ID

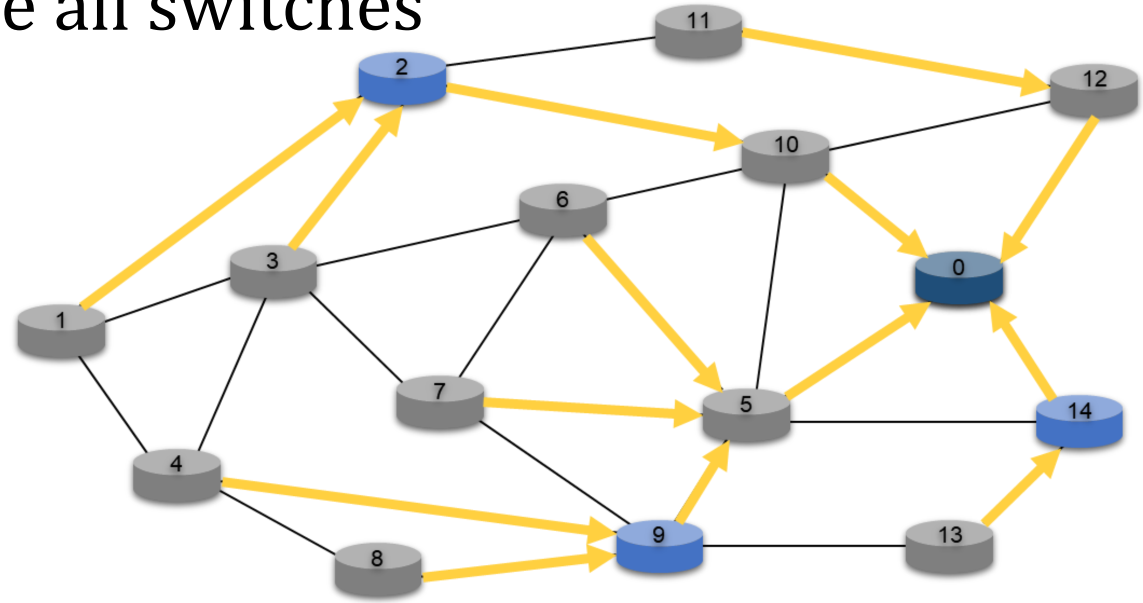
```
0
0  0
1
0  2
2
0  10
3
0  2
4
0  9
5
0  0
6
0  5
```

```
7
0  5
8
0  9
9
0  5
10
0  0
11
0  12
12
0  0
13
0  14
14
0  0
```



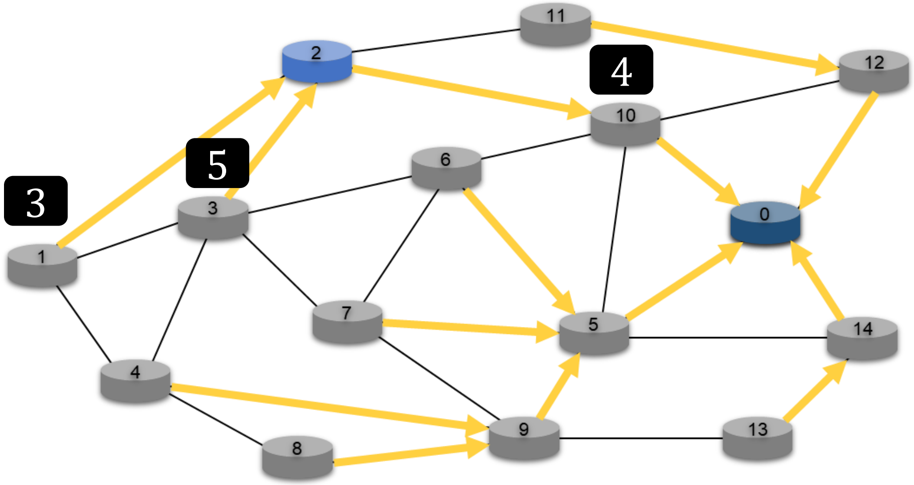
Traditional Switches vs SDN switches

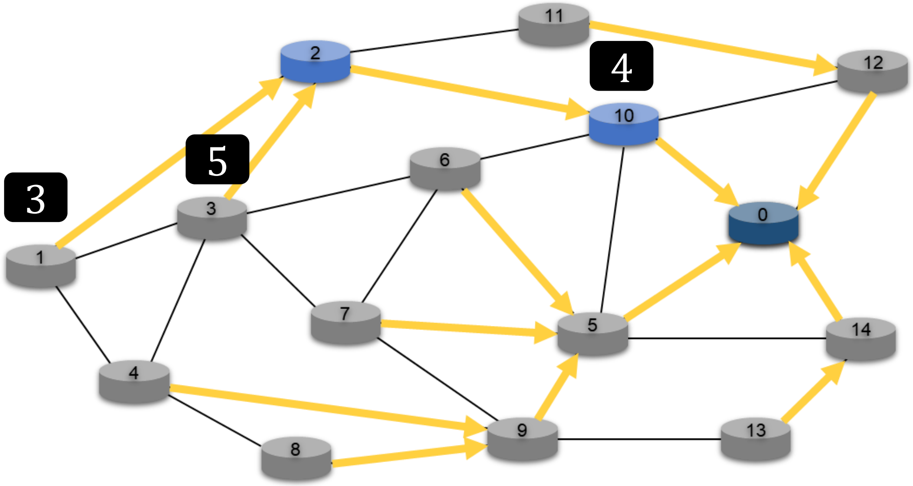
- OSPF paths are fixed → Not flexible
- SDN-enabled switches provide path diversity
- However, SDN switches are expensive
- We cannot upgrade all switches



Which Switch Should be Upgraded First?

- Given a limited budget for upgrading switches
- Each switch has a different upgrade cost
- Goal: minimize the maximum link load





Programming Project #3:
Partially Upgrading an ISP network to SDN

- Input:
  - # nodes, # destinations, # links, and #pairs
  - Nodes (ID) with its upgrade cost to support SDN
  - Destinations (ID)
  - Links between nodes
  - Traffic matrix (flow size for each pair)
  - Limited budget to bound the total upgrade cost of nodes
- Procedure:
  - Compute a set of nodes that will be upgraded to support SDN
  - Compute shortest paths to each destination
  - Compute next hops and portions for SDN-enabled nodes
- Output:
  - The set of SDN-enabled nodes
  - Each node's routing table
  - Packet exchange information will be logged automatically

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 Rules

- Note that you cannot use brute-force algorithm
- Your solution should be deterministic on our server
  - E.g., the random seed & the number of iterations are fixed

Deterministic!

We have a **strict** TIME LIMIT!

Note – Create SDN Nodes and Links

- Define class SDN_switch and Create SDN_switch
  - Derived from class node (Inheritance)
- 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, 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 HW1

Note – Receive and Send Packets (1/2)

- 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() == "TRA_data_packet")
  - if (p->type() == "SDN_ctrl_packet")

Note – Define and Create SDN Controller

- 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 switches 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 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)

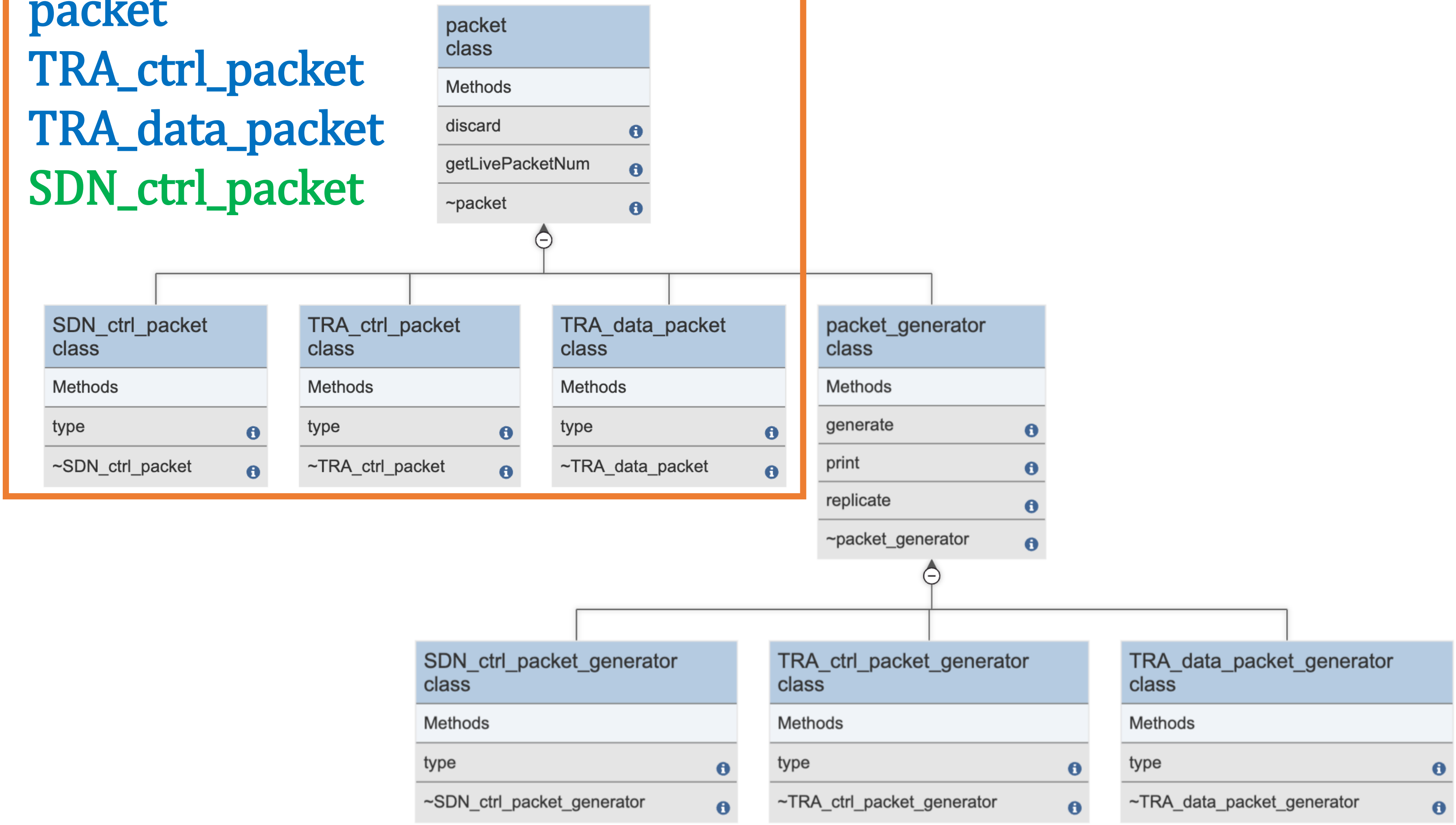
Inheritance

packet

TRA_ctrl_packet

TRA_data_packet

SDN_ctrl_packet



packet class
Methods
discard
getLivePacketNum
~packet

SDN_ctrl_packet class
Methods
type
~SDN_ctrl_packet

TRA_ctrl_packet class
Methods
type
~TRA_ctrl_packet

TRA_data_packet class
Methods
type
~TRA_data_packet

packet_generator class
Methods
generate
print
replicate
~packet_generator

SDN_ctrl_packet_generator class
Methods
type
~SDN_ctrl_packet_generator

TRA_ctrl_packet_generator class
Methods
type
~TRA_ctrl_packet_generator

TRA_data_packet_generator class
Methods
type
~TRA_data_packet_generator

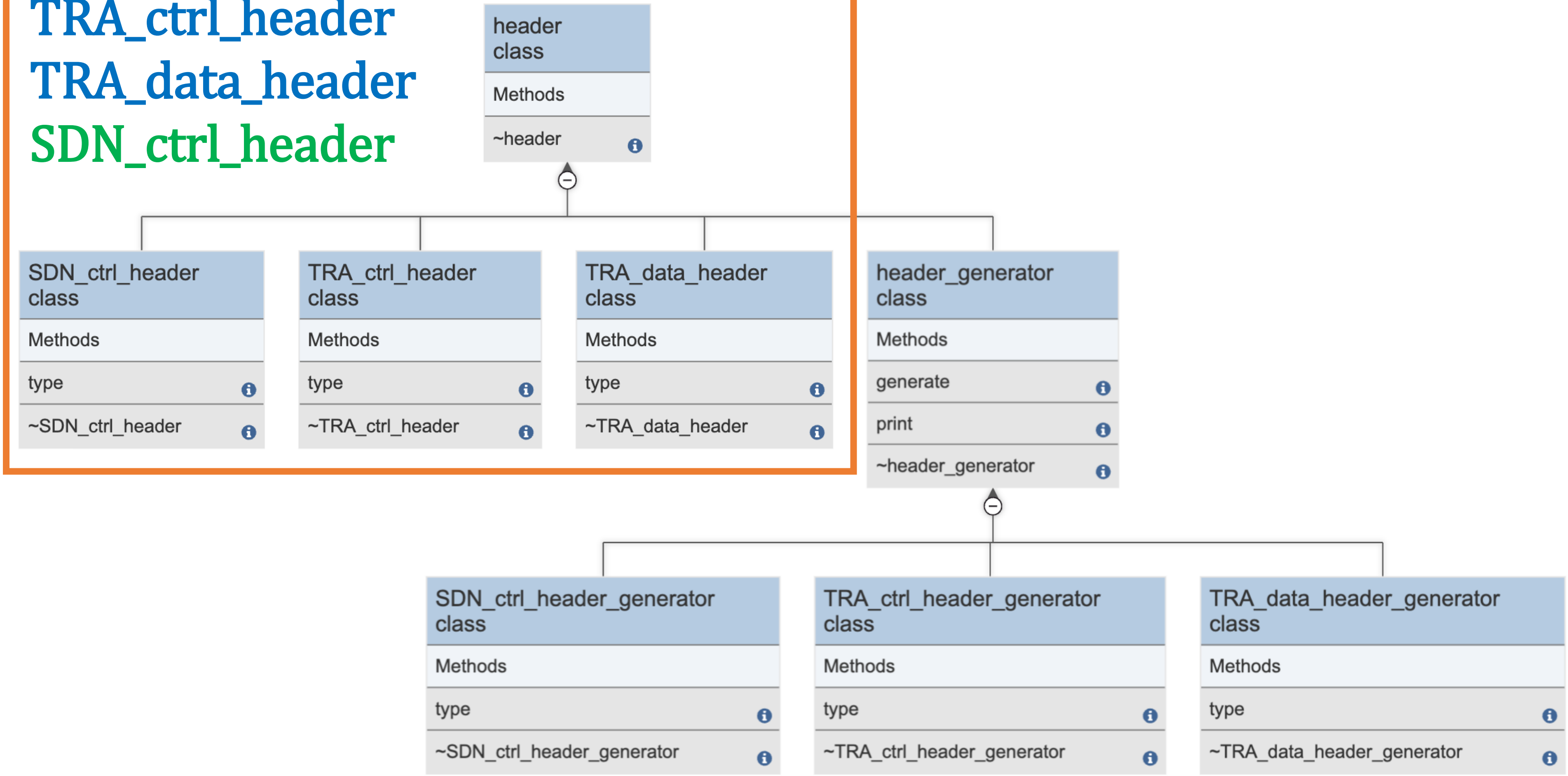
Inheritance

header

TRA_ctrl_header

TRA_data_header

SDN_ctrl_header



header class
Methods
~header

SDN_ctrl_header class
Methods
type
~SDN_ctrl_header

TRA_ctrl_header class
Methods
type
~TRA_ctrl_header

TRA_data_header class
Methods
type
~TRA_data_header

header_generator class
Methods
generate
print
~header_generator

SDN_ctrl_header_generator class
Methods
type
~SDN_ctrl_header_generator

TRA_ctrl_header_generator class
Methods
type
~TRA_ctrl_header_generator

TRA_data_header_generator class
Methods
type
~TRA_data_header_generator

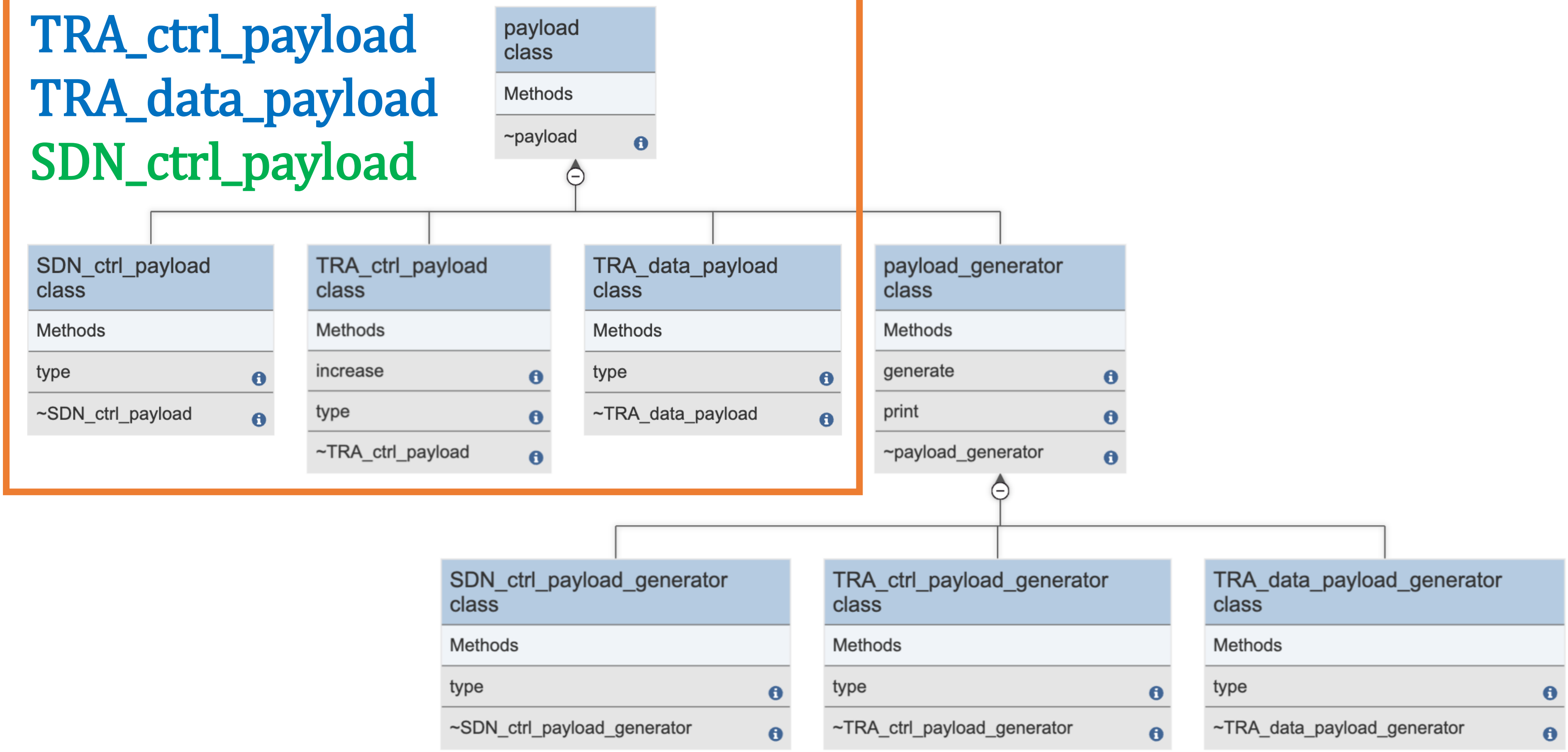
Inheritance

payload

TRA_ctrl_payload

TRA_data_payload

SDN_ctrl_payload



payload
class
Methods
~payload

SDN_ctrl_payload
class
Methods
type
~SDN_ctrl_payload

TRA_ctrl_payload
class
Methods
increase
type
~TRA_ctrl_payload

TRA_data_payload
class
Methods
type
~TRA_data_payload

payload_generator
class
Methods
generate
print
~payload_generator

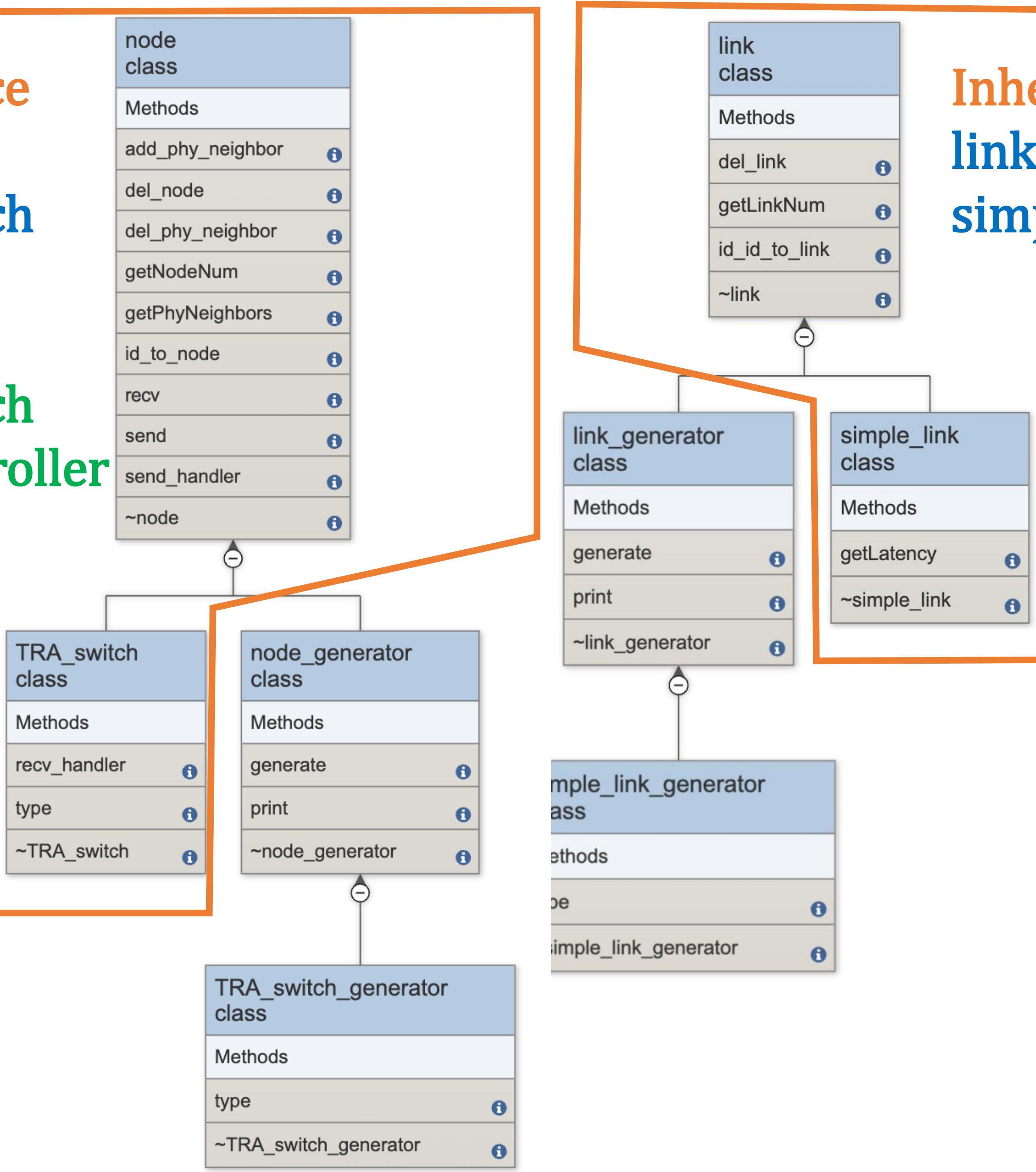
SDN_ctrl_payload_generator
class
Methods
type
~SDN_ctrl_payload_generator

TRA_ctrl_payload_generator
class
Methods
type
~TRA_ctrl_payload_generator

TRA_data_payload_generator
class
Methods
type
~TRA_data_payload_generator

Inheritance
node
TRA_switch

Required:
SDN_switch
SDN_controller

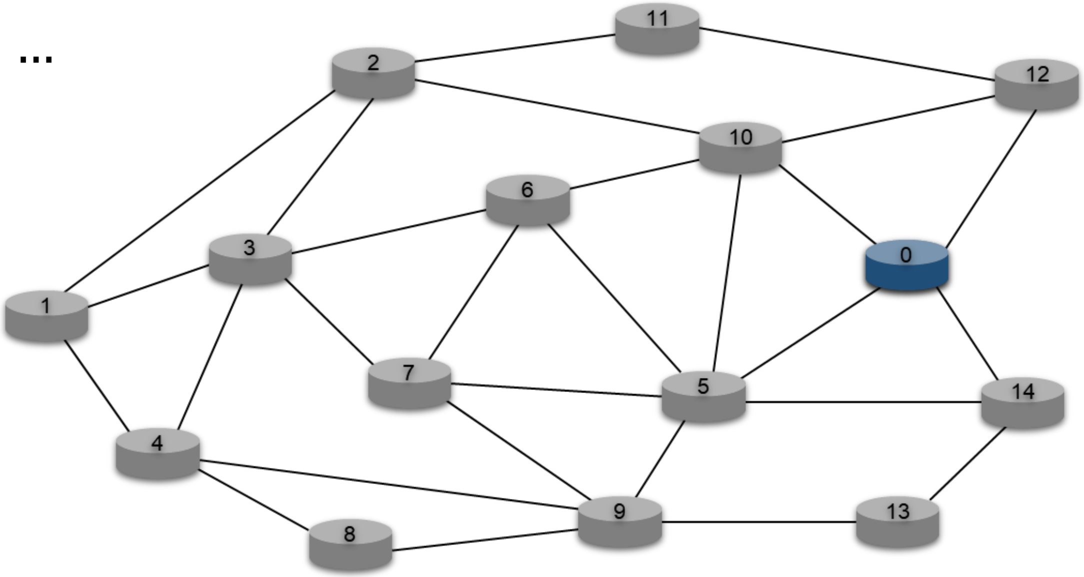


Inheritance
link
simple_link

Input Sample:
use cin

Format:

```
#Nodes  #Dsts  #Links  #Pairs  SimTime  SDN_CtrlTime  Budget
DstID_List
NodeID  UpgradeCost  (BroadcastTime)
...
LinkID  Node1  Node2
...
FlowID  Src  Dst  FlowSize
...
```



```
15  1  28  3  300  200  120
0
0  40  100
1  30
2  40
3  50
4  40
5  60
6  40
7  40
8  20
9  50
10  50
11  20
12  30
13  20
14  30
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

You have to print the set of nodes that will be upgraded and the routing table for each node after event::start_simulate();

Then, you should implement
recv_handler and some components

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

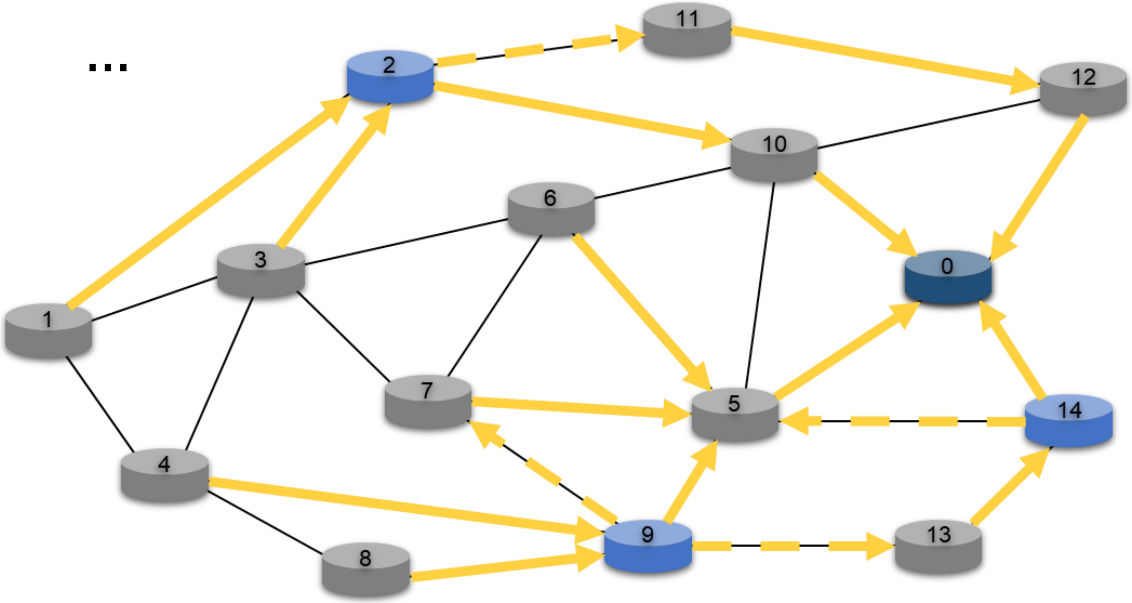
DstID   NextID Portion NextID Portion…

…

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: 5/16 Tue (from 4/18, you have 4 weeks)
- Deadline: 5/30 Tue (from 4/18, you have 6 weeks)
- 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++)**
  - **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