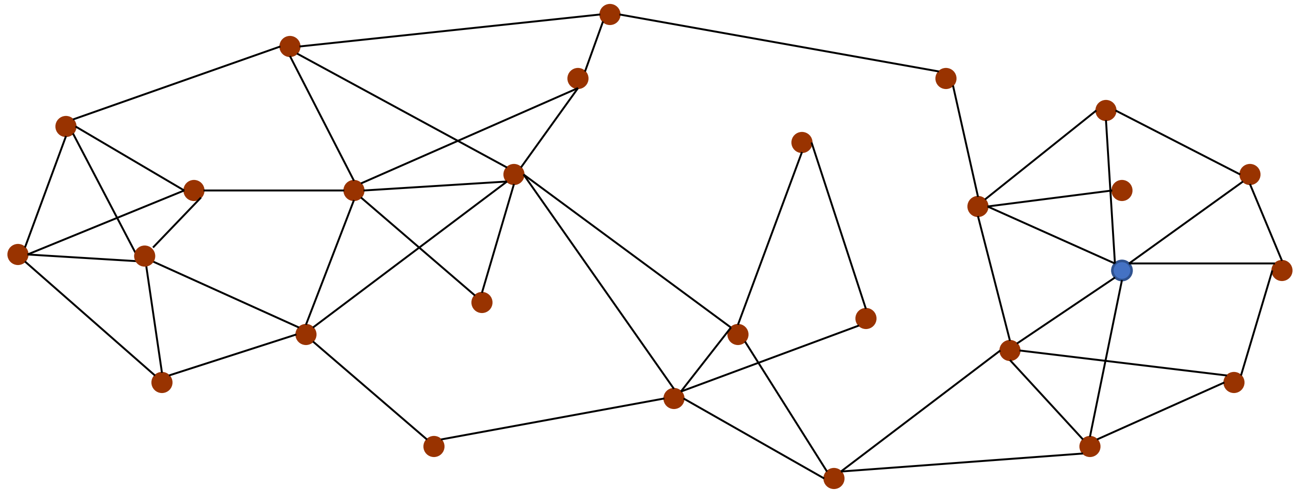
Object-Oriented Programming
Programming Project #2+3

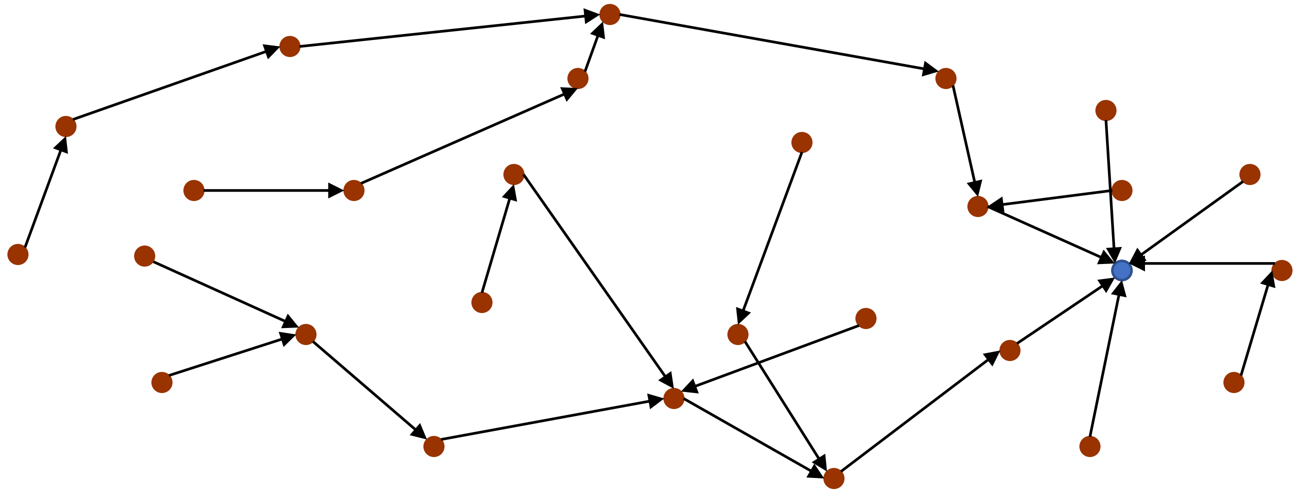
Background

- Large-scale Internet of Things (IoT) or wireless sensor networks
- Every node monitors the environment
- How to collect all the data to the sink? Such as temperature and humidity



Why Aggregation Tree?

- Collect all the data to the sink?
- Limited storage and computation capability
- Routing on a tree → Simple to maintain
- In-network aggregation → Save transmission energy
- Such as min, max, and avg

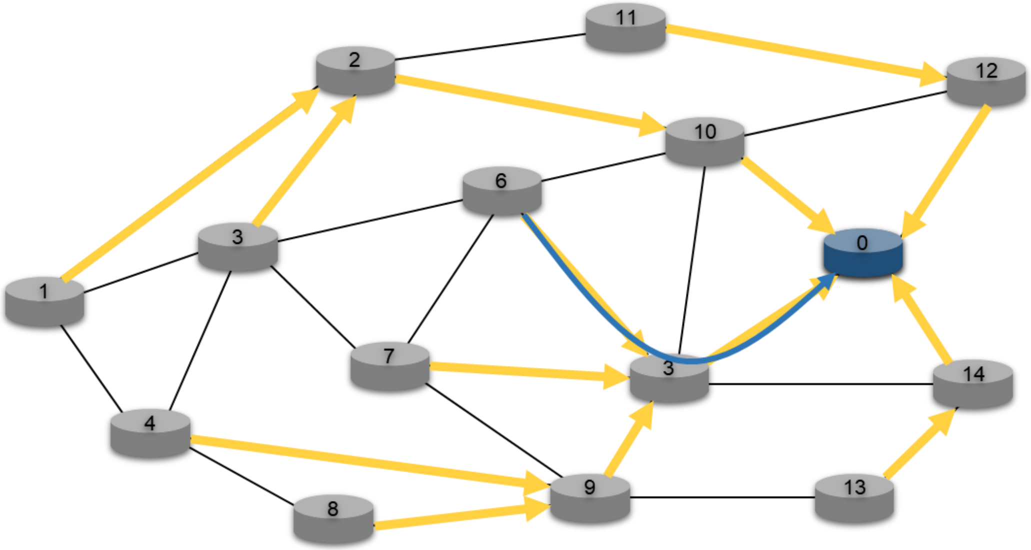


Programming Project #2: Construct an aggregation tree

- Input:
  - A node-weighted network $G = (V, E)$
- Procedure:
  - Construct a tree rooted at node 0 (i.e., sink node)
- Output:
  - Packet exchange information will be logged automatically
  - Every node's parent in the constructed tree

Construct Aggregation Tree

- Use the distributed BFS with IoT_ctrl_packet
- Set the parent to the node has a smaller ID and a smaller hop count to the sink
  - 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 IoT Devices and Links

- Define class IoT_device and Create IoT_device
    - Derived from class node (Inheritance)
- Each node has an unsigned int ID
    - node::node_generator::generate("IoT_device",id);
- Every node only knows its neighbors
- Add the neighbors for each device
    - 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)
- Add an unsigned integer to store the parent of each device in class IoT_device
- Add a vector<unsigned int> to store the children of each device in class IoT_device

Note – Define and Create IoT Sink and Links

- Define new class IoT_sink
  - Derived from class node (Inheritance)
- Before creating the devices, create an IoT Sink
  - sink_id is the sink ID = 0 (before all devices’ ID)
  - node_generator::generate(“IoT_sink", sink_id);
- Connect the IoT sink to its neighbors
  - node::id_to_node(device_id)->add_phy_neighbor(sink_id);
  - node::id_to_node(sink _id)->add_phy_neighbor(device _id);

Note – Generate Data and Ctrl Packets

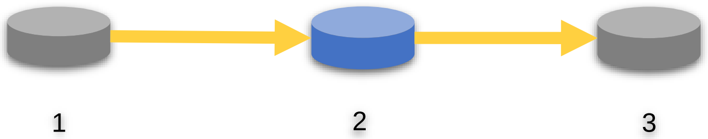
- Generate data packets
  - void **IoT_data_packet_event** (unsigned int src, unsigned int dst=0, unsigned int t = 0, string msg="default")
  - An IoT_data_packet will be generated for a source (src) and sent to a destination (dst) at time t
  - The source (src) will receive the IoT_data_packet first (since it's src)
- Generate ctrl packets
  - void **IoT_ctrl_packet_event** (unsigned int src = 0, unsigned int t = event::getCurTime(), string msg = "default")
  - The function is used to initialize the distributed BFS; that is, a IoT_ctrl_packet will be generated for a source (src) with a counter 0
  - You have to implement recv_handler() in IoT_device 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 parent of every node to build the path from every node to the source
  - The source (src) will receive the IoT_ctrl_packet first (since it's src)

Note – Receive and Send Packets (1/2)

- Define the rules to handle the received packet in class IoT_device's member function recv_handler
  - void IoT_device::recv_handler (packet *p)
  - Don't use node::id_to_node(id) in recv_handler
- Get the current device's ID and its neighbor
  - Use getNodeID() in recv_handler
  - Use getPhyNeighbors().find(n_id) to check whether the IoT device 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() == "IoT_data_packet")
  - if (p->type() == "IoT_ctrl_packet")

Note – Receive and Send Packets (2/2)

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



PreID = 1
NexID = 2

PreID = 2
NexID = 3

Inheritance

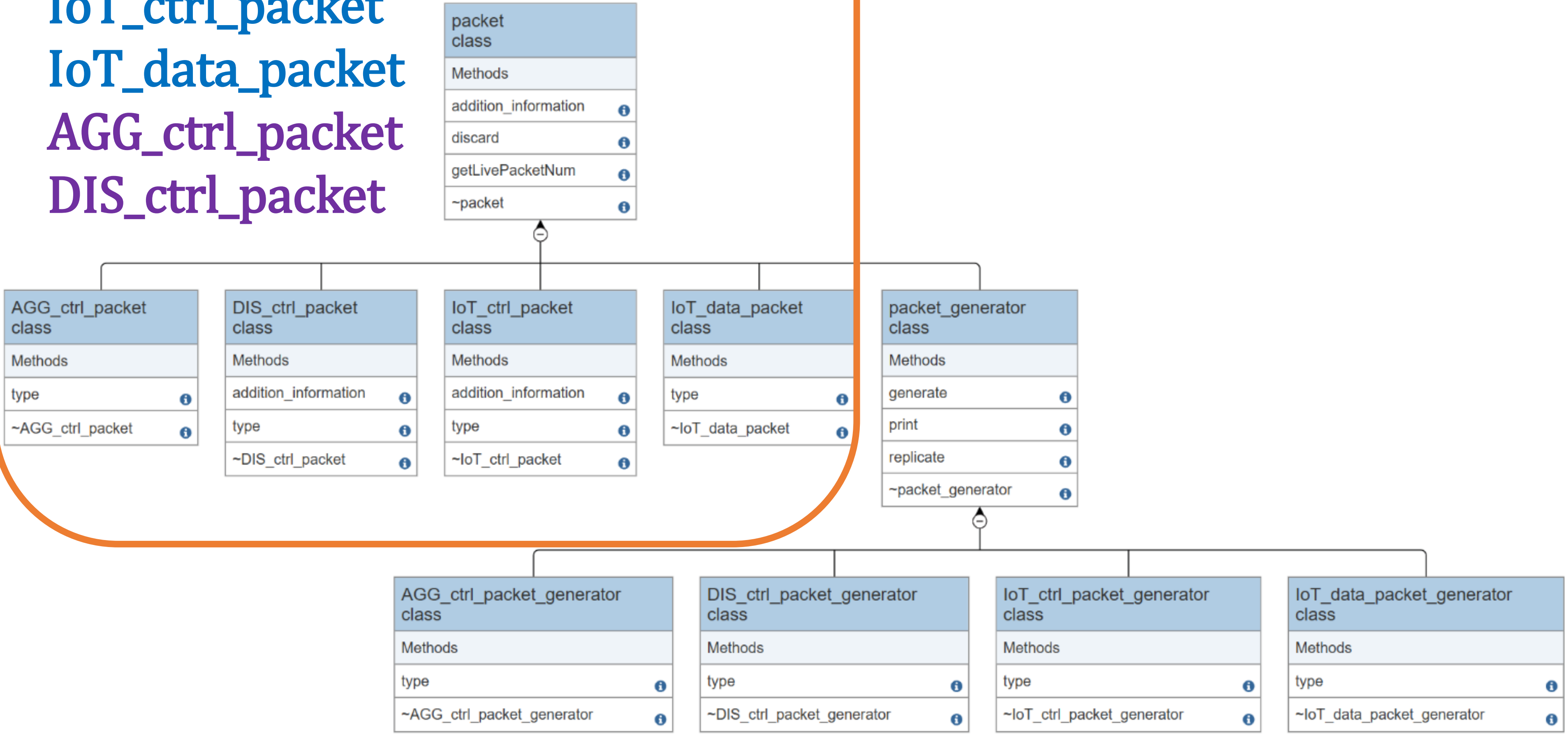
packet

IoT_ctrl_packet

IoT_data_packet

AGG_ctrl_packet

DIS_ctrl_packet



packet class
- Methods
- addition_information
- discard
- getLivePacketNum
- ~packet

AGG_ctrl_packet class
- Methods
- type
- ~AGG_ctrl_packet

DIS_ctrl_packet class
- Methods
- addition_information
- type
- ~DIS_ctrl_packet

IoT_ctrl_packet class
- Methods
- addition_information
- type
- ~IoT_ctrl_packet

IoT_data_packet class
- Methods
- type
- ~IoT_data_packet

packet_generator class
- Methods
- generate
- print
- replicate
- ~packet_generator

AGG_ctrl_packet_generator class
- Methods
- type
- ~AGG_ctrl_packet_generator

DIS_ctrl_packet_generator class
- Methods
- type
- ~DIS_ctrl_packet_generator

IoT_ctrl_packet_generator class
- Methods
- type
- ~IoT_ctrl_packet_generator

IoT_data_packet_generator class
- Methods
- type
- ~IoT_data_packet_generator

Inheritance

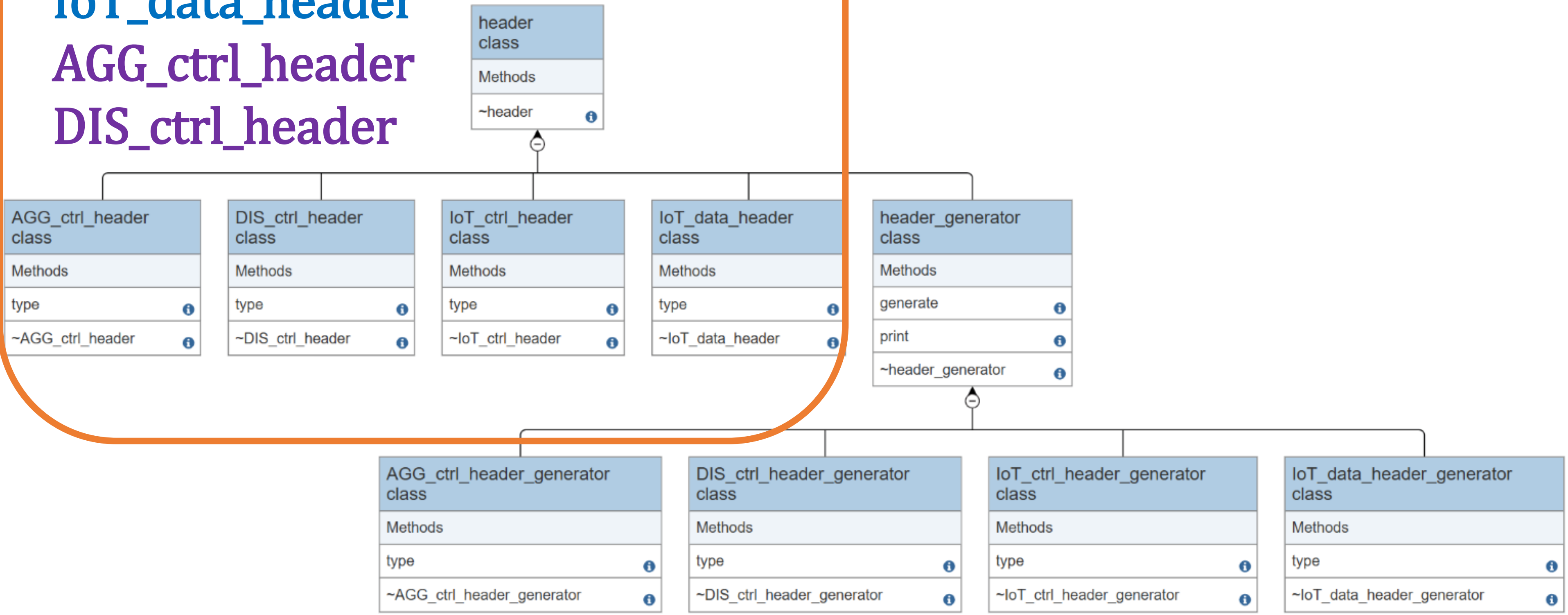
header

IoT_ctrl_header

IoT_data_header

AGG_ctrl_header

DIS_ctrl_header



Inheritance

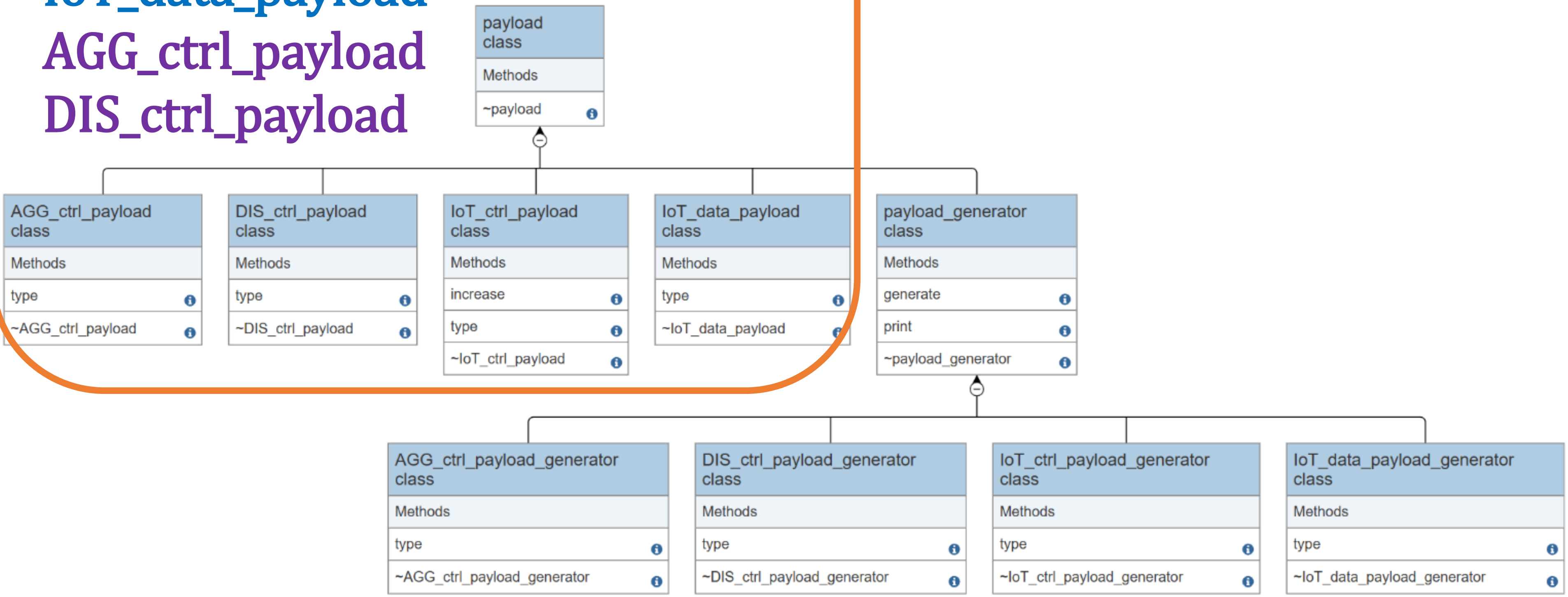
payload

IoT_ctrl_payload

IoT_data_payload

AGG_ctrl_payload

DIS_ctrl_payload



payload class
- Methods
- ~payload

AGG_ctrl_payload class
- Methods
- type
- ~AGG_ctrl_payload

DIS_ctrl_payload class
- Methods
- type
- ~DIS_ctrl_payload

IoT_ctrl_payload class
- Methods
- increase
- type
- ~IoT_ctrl_payload

IoT_data_payload class
- Methods
- type
- ~IoT_data_payload

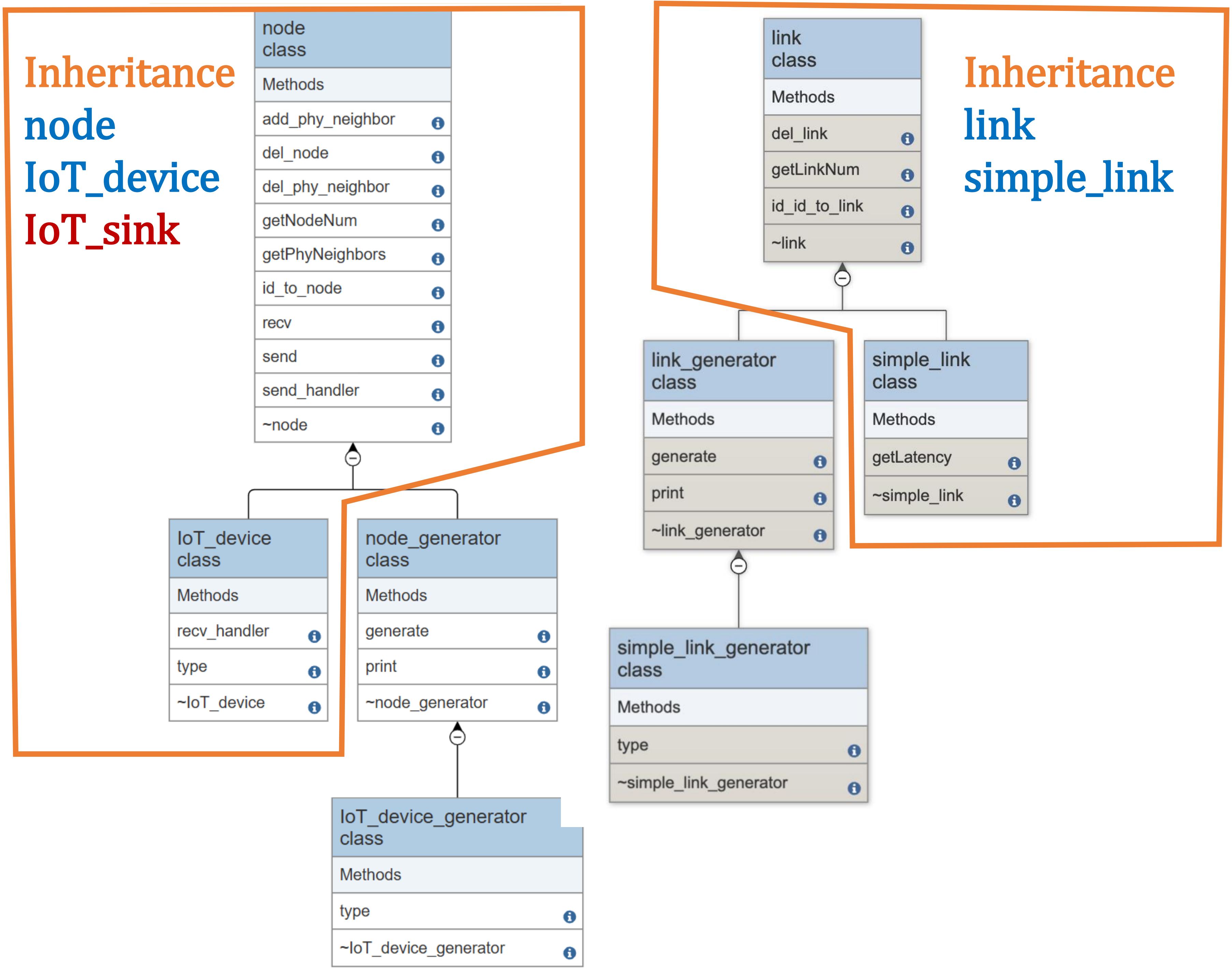
payload_generator class
- Methods
- generate
- print
- ~payload_generator

AGG_ctrl_payload_generator class
- Methods
- type
- ~AGG_ctrl_payload_generator

DIS_ctrl_payload_generator class
- Methods
- type
- ~DIS_ctrl_payload_generator

IoT_ctrl_payload_generator class
- Methods
- type
- ~IoT_ctrl_payload_generator

IoT_data_payload_generator class
- Methods
- type
- ~IoT_data_payload_generator



Inheritance
node
IoT_device
IoT_sink

node class

Methods

- add_phy_neighbor
- del_node
- del_phy_neighbor
- getNodeNum
- getPhyNeighbors
- id_to_node
- recv
- send
- send_handler
- ~node

IoT_device class

Methods

- recv_handler
- type
- ~IoT_device

node_generator class

Methods

- generate
- print
- ~node_generator

IoT_device_generator class

Methods

- type
- ~IoT_device_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

Overview

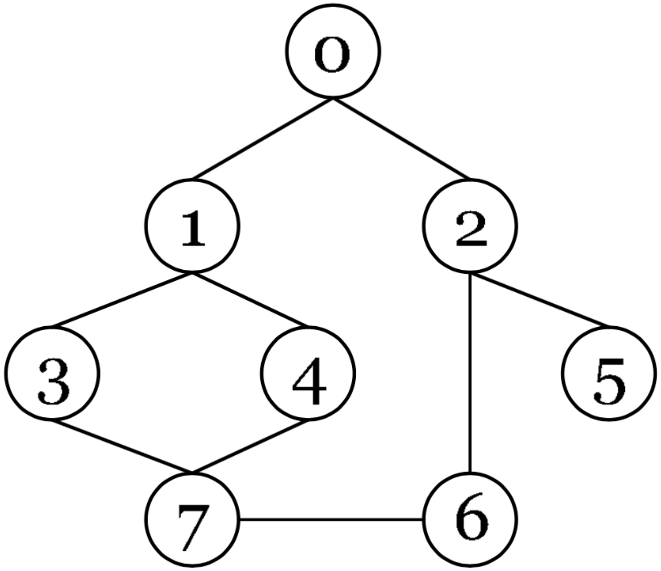
- The sink (node 0) receives the IoT_ctrl_packet and then starts the distributed BFS
  - Each node receives the IoT_ctrl_packet to know its parent and children on the aggregation tree
- Each device obtains its IoT_data_packet at the data transmission time
  - Each leaf device (i.e., each node having no child) directly sends the packet to its parent once it obtains the data packet
  - Each non-leaf non-sink device sends its data packet to its parent after it receives all the packets from all its children; please use send_handler in the non-leaf non-sink node to send the packet once a receiving all the packets from all its children

Input Sample:

use cin

Format:

```
#Nodes   #Links
SimTime   BFS_Start_Time   Data_Trans_Time
Link_ID   Link_End1   Link_End2
...
```



Example:

```
8     9
300   0     100
0     0     1
1     0     2
2     1     3
3     1     4
4     2     5
5     2     6
6     3     7
7     4     7
8     6     7
```

Output Sample:
use cout

You have to print the parents after event::start_simulate();

The remaining output will be automatically generated ☺

Note that the output could be different in different computers

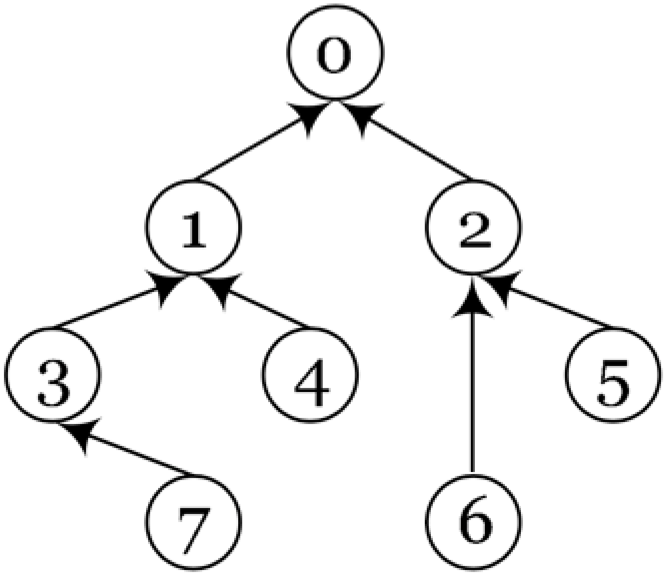
Output Sample (continue):
use cout

Format:

- The automatic printing (you can’t change)

NodeID NextID

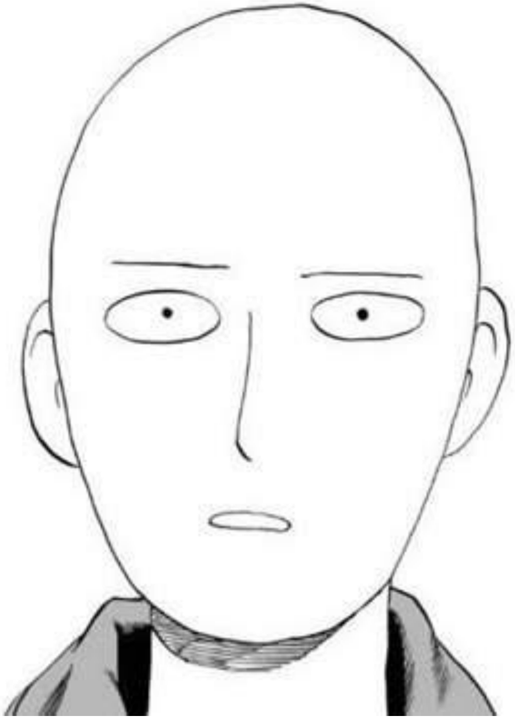
...



Example:

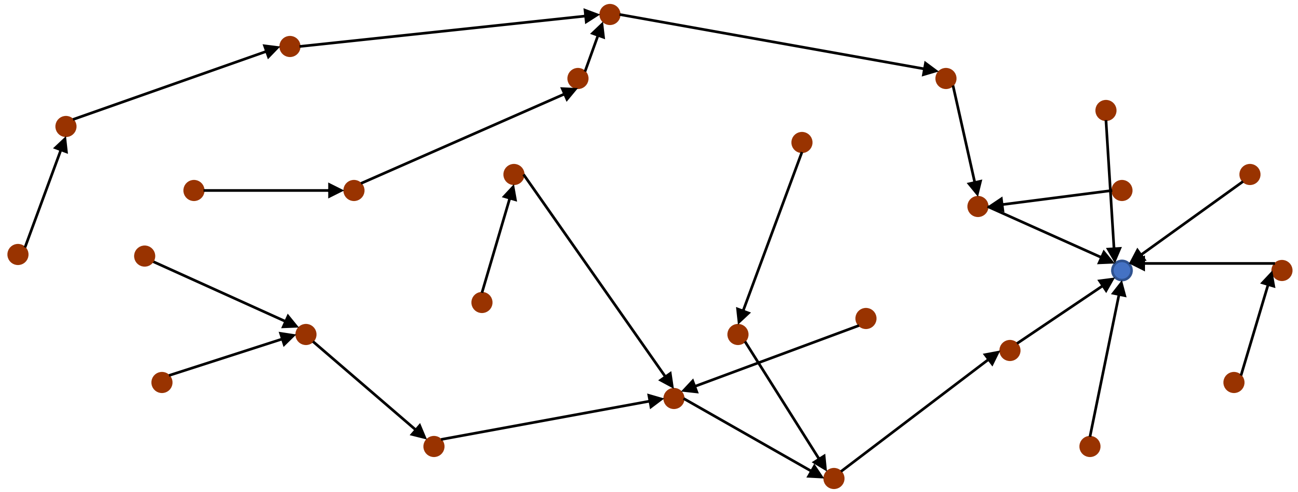
```
0    0
1    0
2    0
3    1
4    1
5    2
6    2
7    3
```

Its own ID



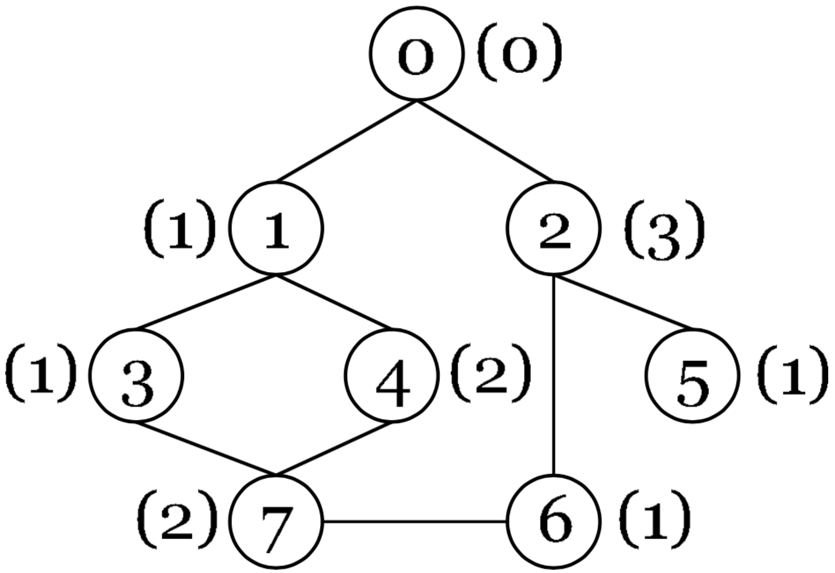
Advanced Data Aggregation

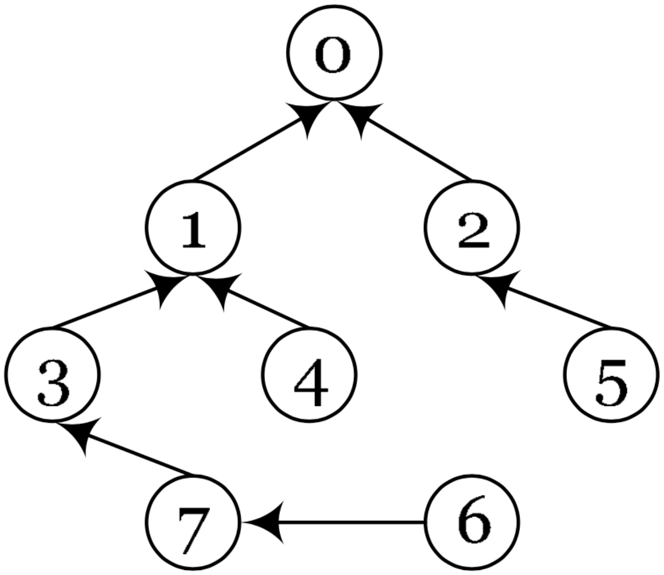
- Some scenarios disallow in-network aggregation
- Alternative way:
  Aggregate the data with a specific total size into one packet to save transmission energy
- See the examples in the following slides



Example 1

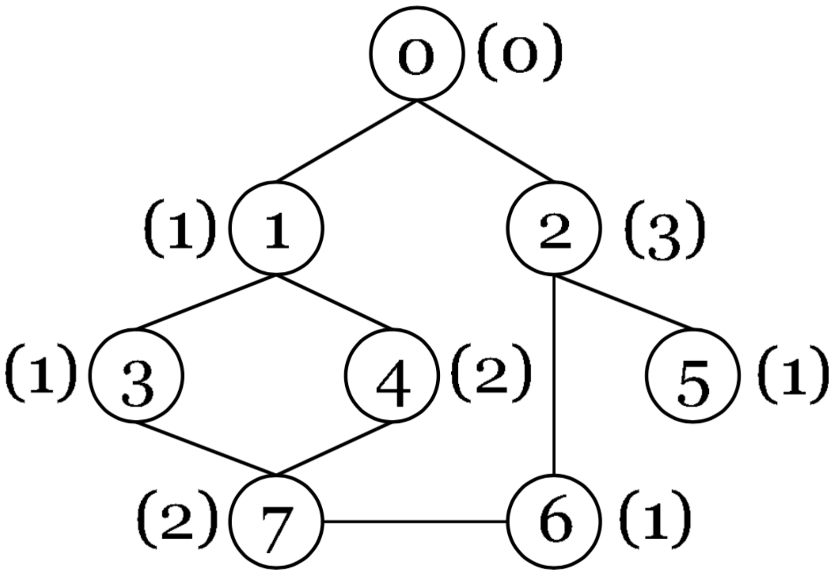
- Routing on a tree → Simple to maintain
- Data aggregation → Save transmission energy
- Nodes' distinct data size (e.g., 1-5)
- Fixed packet size (e.g., 3)



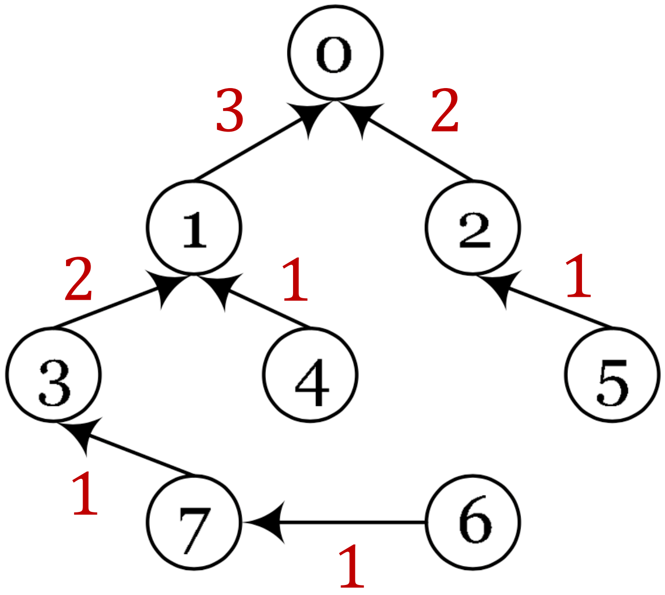


Example 1

- Routing on a tree → Simple to maintain
- Data aggregation → Save transmission energy
- Nodes' distinct data size (e.g., 1-5)
- Fixed packet size (e.g., 3)

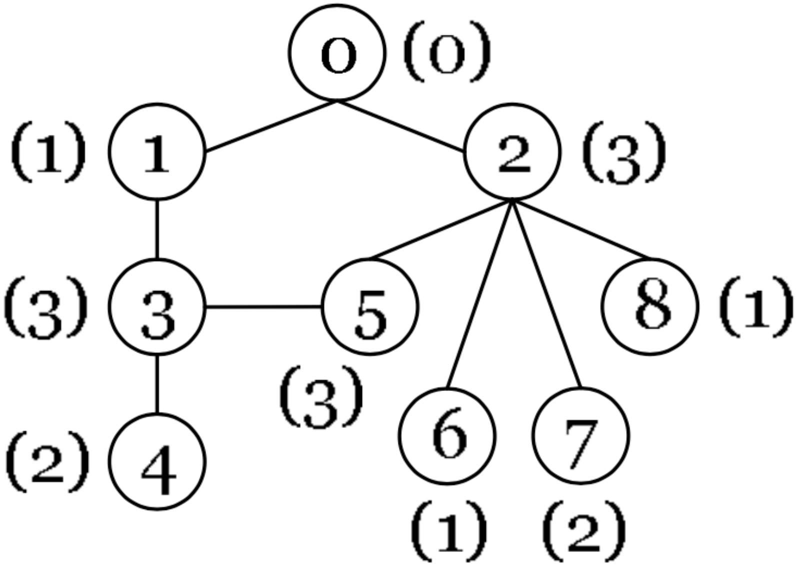


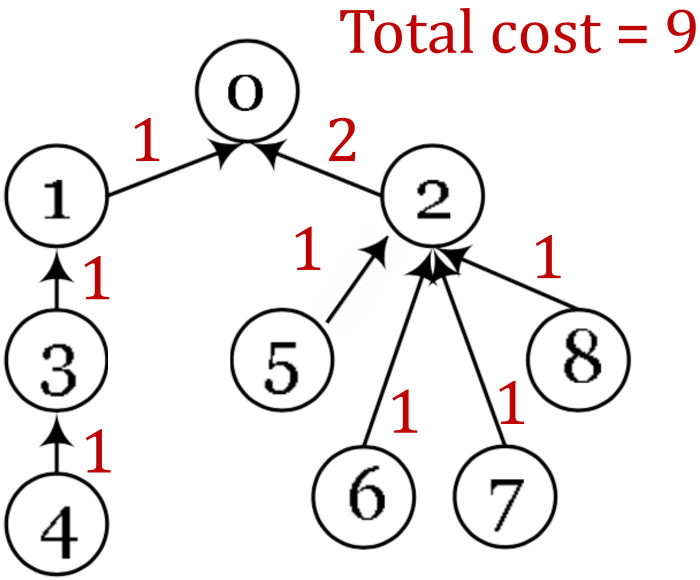
Total cost = 11



Example 2

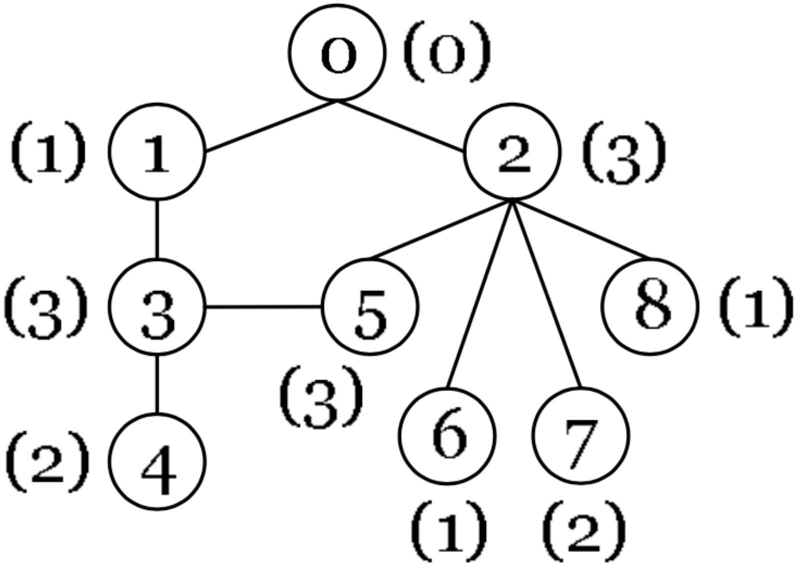
- Routing on a tree → Simple to maintain
- Data aggregation → Save transmission energy
- Nodes’ distinct data size (e.g., 1-5)
- Fixed packet size (e.g., 9)



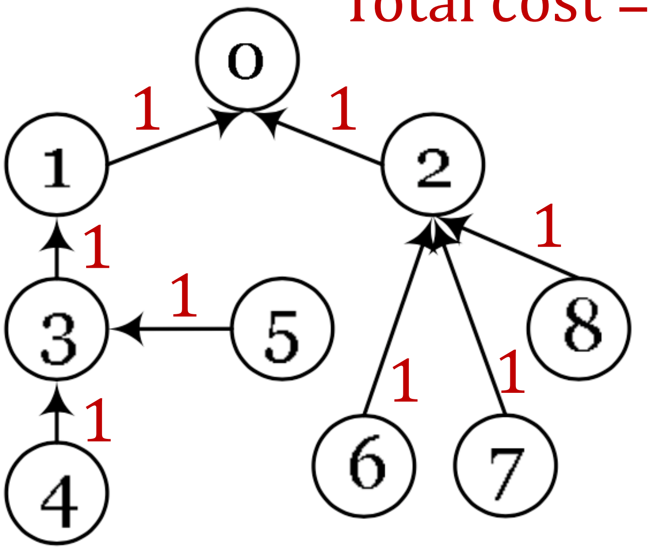


Example 2

- Routing on a tree → Simple to maintain
- Data aggregation → Save transmission energy
- Nodes' distinct data size (e.g., 1-5)
- Fixed packet size (e.g., 9)



Total cost = 8



Bad News

- The problem is NP-hard
- We may not always find the optimal solution in polynomial time
- Alternatively, we aim at a near-optimal solution

Programming Project #3: Construct an advanced aggregation tree

- Input:
  - A node-weighted network $G = (V, E)$
  - Packet size
  - Data size of each node
- Procedure:
  - Construct a tree rooted at node 0 (i.e., sink node)
- Output:
  - Packet exchange information will be logged automatically
  - Every node's parent in the constructed tree

The Competition

- The grade is inversely proportional to the total cost
- Basic: 60 (deadline)
  - Feasible solution
- 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

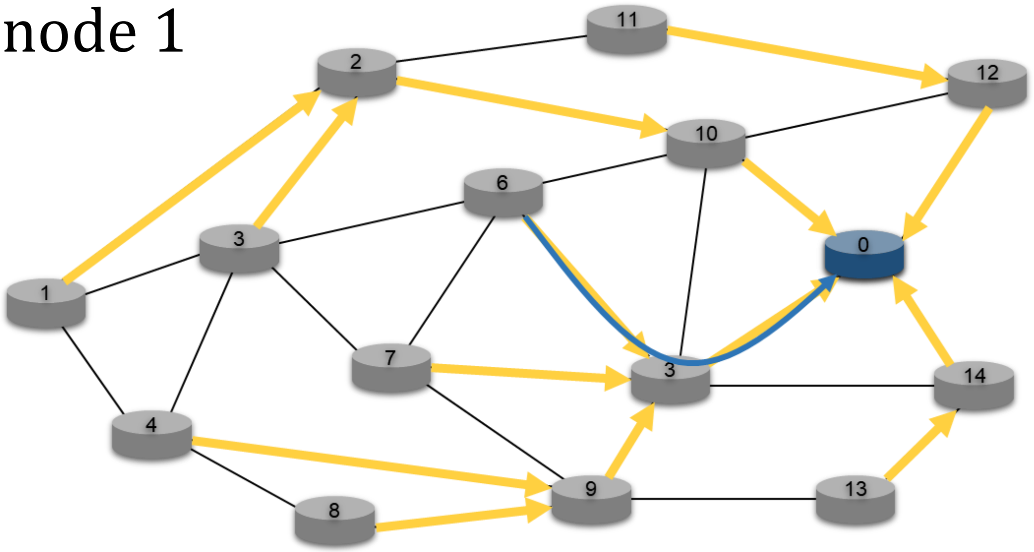
- Your solution should be deterministic on our server
  - E.g., the random seed & the number of iterations are fixed

Deterministic!

We have a **strict** TIME LIMIT!

Note – Create SDN Nodes and Links

- For class IoT_device:
  - Add a map<unsigned int, unsigned int> to store the reverse next hop for each descendant of each device
- For example:
  - The path: 1 -> 2 -> 10 -> 0
  - The reverse path: 0 -> 10 -> 2 ->1
  - Node 10 will mark node 2 as the reverse next hop toward node 1



Note

- More ctrl packets (1/2)
  - void **AGG_ctrl_packet_event** (unsigned int src, unsigned int dst=0, unsigned int t = event::getCurTime(), string msg="default")
  - An AGG_data_packet will be generated for a source (src) and sent to a destination (dst) at time t
  - The source (src) will receive the AGG_data_packet first (since it’s src)
  - The AGG_ctrl_packet is used to send the neighbor list from each node to the sink
  - The packet is sent to the parent iteratively until it reaches the sink; Relaying each child’s packet separately
  - The intermediate node receives the packet should record the reverse path

Note

- More ctrl packets (2/2)
  - void **DIS_ctrl_packet_event** (unsigned int sink_id = 0, unsigned int t = event::getCurTime(), string msg = "default")
  - A packet will be generated for notifying the sink to send the DIS_ctrl_packet to every descendant node at time t (optional) to update the node's parent
  - Before sending the packet to each node, the sink's recv_handler should use setParent() to set the packet's column parent to the new parent
  - setParent(unsigned int): set the new parent of the node
  - Each node can use getParent() of DIS_ctrl_payload to get parent from the DIS_ctrl_packet
  - The sink will receive the DIS_ctrl_packet first
    (since the sink is src)

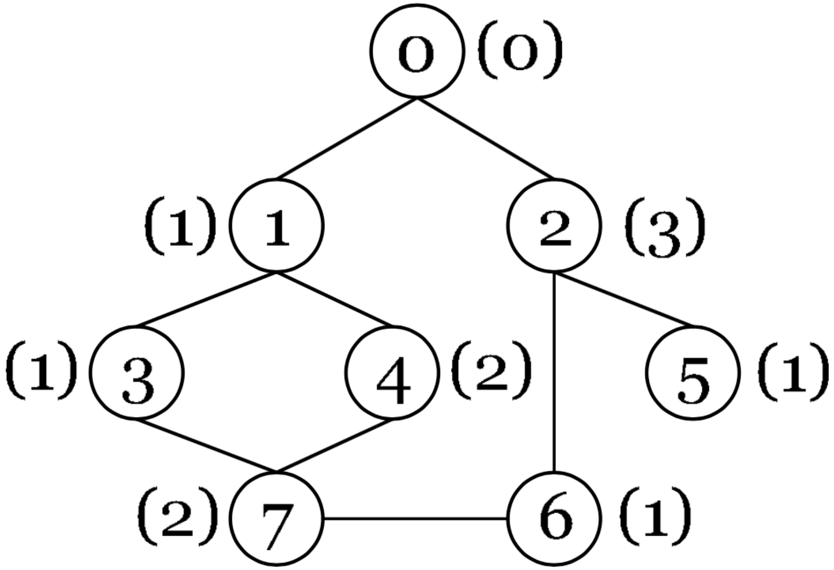
Overview

- The sink (node 0) receives the IoT_ctrl_packet and then starts the distributed BFS at BFS_Start_Time
  - Each node receives the IoT_ctrl_packet to know its parent and children on the original aggregation tree
- Each device obtains its AGG_ctrl_packet at AGG_Start_Time
  - Each non-sink node sends its neighbor list (stored in the **msg**) to the sink; please use send_handler in the non-sink node to send the packet to the sink
  - Each intermediate node records the reverse path
- The sink calculates each node's new parent at DIS_Start_time and then sends the new parent information back to each node
- Each device starts to send the IoT_data_packet to the sink via the new parent at Data_Trans_Time; For simplicity, you only need to send a packet to its parent

Input Sample:
use cin

Format:

#Nodes #Links Packet_Size
SimTime BFS_Start_Time AGG_Start_Time
DIS_Start_Time Data_Trans_Time
Node_ID Data_size
...
Link_ID Link_End1 Link_End2
...



Example:

```
8      9      3
300    0      50
100    200
0      0
1      1
2      3
3      1
4      2
5      1
6      1
7      2
0      0      1
1      0      2
2      1      3
3      1      4
4      2      5
5      2      6
6      3      7
7      4      7
8      6      7
```

Output Sample:
use cout

You have to print the new parents after event::start_simulate();

The remaining output will be automatically generated ☺

Note that the output could be different in different computers

Output Sample (continue):
use cout

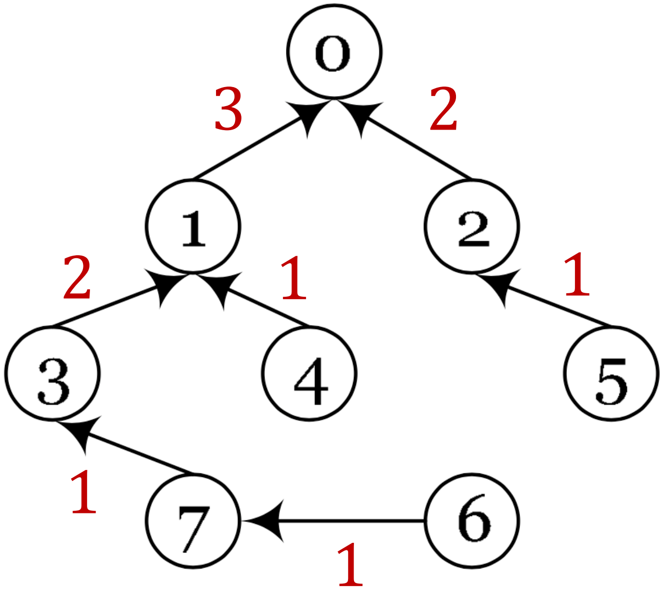
Format:

- The automatic printing (you can't change)

NodeID   NextID

...

Total cost = 11



Example:

Its own ID

0	0
1	0
2	0
3	1
4	1
5	2
6	7
7	3

Note

- Superb deadline: 6/6 Thu (from 5/9, you have 4 weeks)
- Deadline: 6/20 Thu (from 5/9, 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