Cairo University
Faculty of Engineering
Computer Department

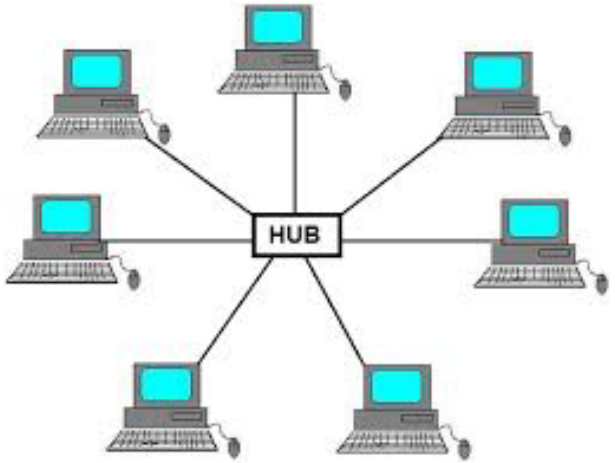
Computer Networks-1
CMP405-A /CMPN405
Fall 2023

Network topologies with omnet++

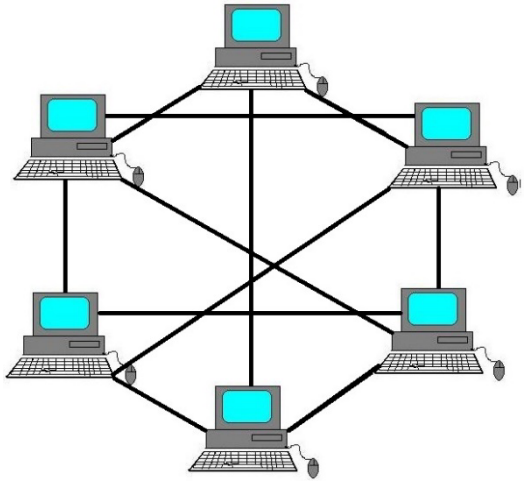
Today's goal

- Build a LAN of an arbitrary N nodes.
- Any 2 nodes can send messages to each other (directly or indirectly).
- Experiment with different network topologies

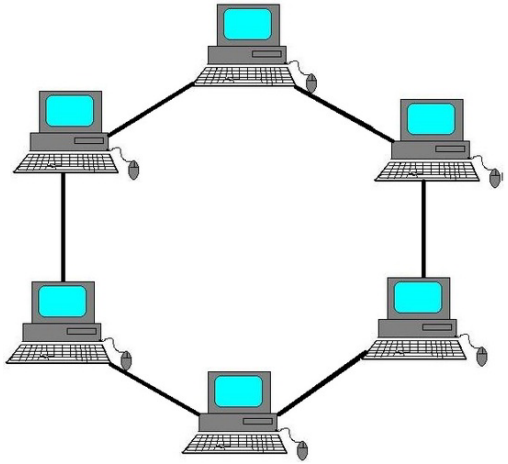
Star



Mesh



Ring



We'll learn about

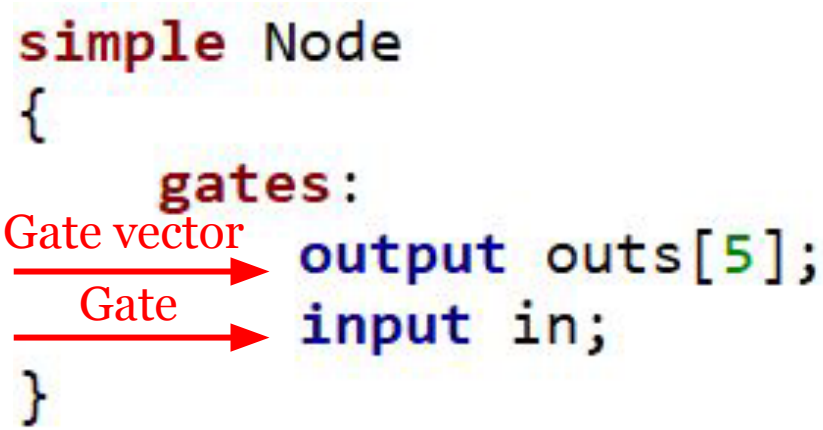
1. Gate vectors
2. Dynamic connections
3. Sampling from random distributions
4. Self messaging

Gate vectors

Building gates of different sizes

Gate vectors

- We can create single gates and gate vectors.
- Gate vectors follow the square brackets syntax “[]”
- The size of a gate vector can be specified or left empty.
- If left empty, the size is inferred from the connections, or is determined when instantiated as a submodule.
- You can find out the size of a gate by calling *gateSize(<gate_name>)*




```
gateSize("outs"); //Returns 5
```

If size is left empty

Size is inferred

```
simple Node
{
    gates:
        inout port[];
}

network Network
{
    submodules:
        node1: Node;
        node2: Node;
        node3: Node;
    connections:
        node1.port++ <--> node2.port++;
        node1.port++ <--> node3.port++;

}
```

Node1 has gate size 2, nodes 2 & 3 have a size of 1

Size is specified when instantiated

```
network Network
{
    submodules:
        node1: Node {
            gates:
                port[2];
        }
        node2: Node {
            gates:
                port[1];
        }
        node3: Node {
            gates:
                port[1];
        }
    connections:
        node1.port++ <--> node2.port++;
        node1.port++ <--> node3.port++;
}
```

Gate vectors cont.

- Gate vectors can accommodate and expand to grow their pre-determined size.
- This code runs properly without any errors.
- All nodes have a gate size of 2

```
network Network
{
    submodules:
        node1: Node {
            gates:
                port[2];
        }
        node2: Node {
            gates:
                port[1];
        }
        node3: Node {
            gates:
                port[1];
        }
    connections:
        node1.port++ <--> node2.port++;
        node1.port++ <--> node3.port++;
        node2.port++ <--> node3.port++;
}
```

Gate vectors

- Use the square brackets syntax.
- Used with the “++” operator
- This automatically expands the size of our gate accordingly.

```
simple Node
{
    gates:
        inout port[];
}

network Network
{
    submodules:
        node1: Node;
        node2: Node;
        node3: Node;
    connections:
        node1.port++ <--> node2.port++;
        node1.port++ <--> node3.port++;
        node2.port++ <--> node3.port++;
}
```

Gate vectors cont.

- Omnetpp by default doesn't allow for a gate to be unconnected.
- An error will be produced when attempting to run with a gate that is unconnected.
- We can work around this by using the *@loose* property

```
simple GridNode {
    gates:
        inout neighbour[4] @loose;
}
```

Gate vectors cont.

We can also specify the size by sub-classing.

```
simple TreeNode {
    gates:
        inout parent;
        inout children[];
}

simple BinaryTreeNode extends TreeNode {
    gates:
        children[2];
}
```

Dynamic connections

Dealing with dynamic networks

- We can use “for loops” and “if conditions” inside *connections* to deal with arbitrarily large modules and add some logic to our topology.

```
network Network
{
    parameters:
        int N = default(4);
    submodules:
        nodes[N]: Node;
    connections:

}
```

??

For loops and if statements in ned (1)

- If statement

```
if p>0 {
    a.out --> b.in;
    a.in <-- b.out;
}
```

For loops and if statements in ned (2)

- For loop

```
for i = 0..count-2 {
        node[i].port[1] <--> node[i+1].port[0];
}
```

For loops and if statements in ned (3)

- For loop and if statement inside it

```
for i=0..sizeof(c)-1, if i%2==0 {
    c[i].out --> out[i];
    c[i].in <-- in[i];
}
```

For loops and if statements in ned(4)

- One line for loop and if statement inside it

```
a.out --> b.in;
c.out --> d.in if p>0;
e.out[i] --> f[i].in for i=0..sizeof(f)-1, if i%2==0;
```

For loops and if statements in ned (5)

- Nested loops and if statement inside it

```
for i=0..sizeof(e)-1, for j=0..sizeof(e)-1 {
    e[i].out[j] --> e[j].in[i] if i!=j;
}
```

For loops and if statements in ned (5)

- Nested loops and if statement inside it “another way to write it”

```
for i=0..sizeof(d)-1, for j=0..sizeof(d)-1, if i!=j {
    d[i].out[j] --> d[j].in[i];
}
```

Dealing with dynamic networks

- We can use for loops and if conditions inside *connections* to deal with arbitrarily large modules and add some logic to our topology.

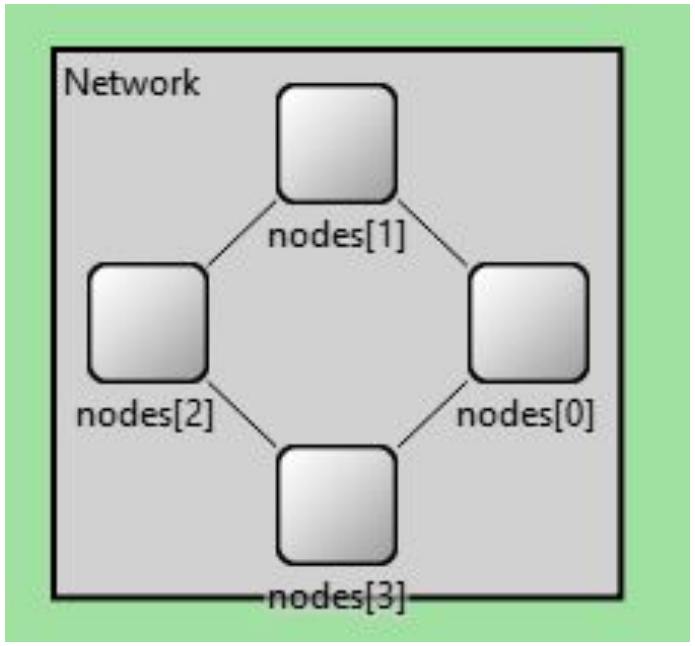
```
network Network
{
    parameters:
        int N = default(4);
    submodules:
        nodes[N]: Node;
    connections:
        for i=0..N-1 {
            nodes[i].port++ <--> nodes[i+1].port++ if i < N-1;
            nodes[i].port++ <--> nodes[0].port++ if i == N-1;
        }
}
```

Which topology is this?

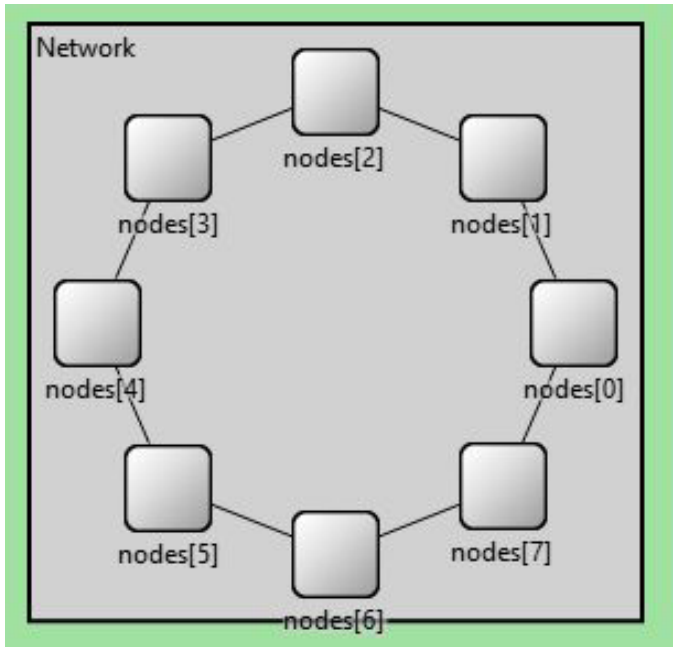
Dynamic ring network

We put N as a parameter “defined in the Network system module and initialized in the .ini file.

N=4 (default)



N=8



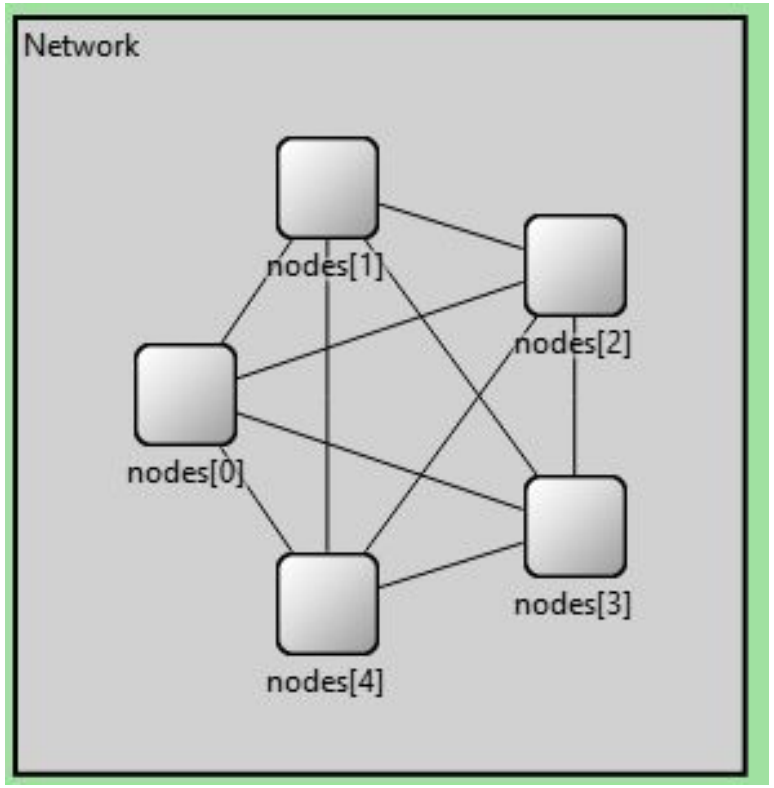
Nested loops

```
connections:
    for i=0..N-1, for j=i+1..N-1
    {
        nodes[i].port++ <--> nodes[j].port++;
    }
```

Which topology is this?

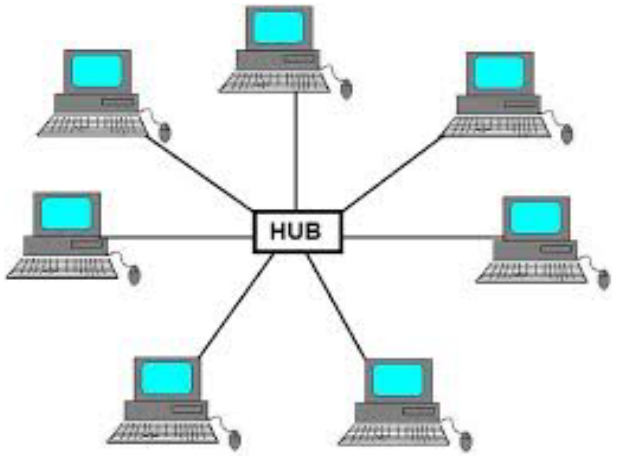
Mesh topology

```
connections:
    for i=0..N-1, for j=i+1..N-1
    {
        nodes[i].port++ <--> nodes[j].port++;
    }
```



Today's lab:

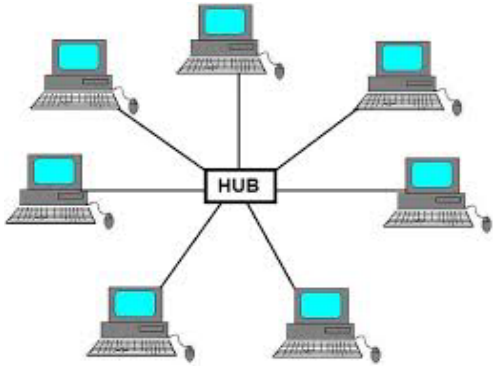
What about the star topology ?



Star

Today's lab requirement number 1/2:

- Create a Star network topology consisting of one Hub and N nodes.
- N should be configurable in the .ini file.
- Each node is connected only to the hub.
- Before starting, every node prints to the console a message " Starting Node i", where i is the node index.
- The hub starts by sending to a random node "m" a message "Hello from Hub".
- The node "m" is selected randomly.
- The node "m" should respond with "Hi from node" to the Hub.
- After a random time, the Hub selects another random node "m" to send the same message to it.
- The node responds with the same response …. etc.



Star

How to do this?

Exponential function

- We want the Hub to send messages to the nodes after a random time from receiving a triggering message.
- We can use the function e*xponential* to do this.

```
double interval=  exponential(2.0)
```

- We need to send ourselves a message in the future, to trigger the sending at the appropriate time.
- We treat this like a self timer and so we use the self messages in ned.

Self messaging

- The *scheduleAt(t, msg)* function allows us to send ourselves a message that will be received at simulation time *t*.
- The *simTime()* functions returns the current simulation time.
- We can combine this with our samples interval to send ourselves a message as follows:

```
scheduleAt(simTime() + interval, new cMessage(""));
```

Self messaging cont.

- Then, inside the *handleMessage(cMessage *msg)* function, we can check if this is a self message or not using the *isSelfMessage()* function

```
if (msg->isSelfMessage()) {
    //This is a self message. Create and send a new packet.
}
else {
    //This msg was sent by another node.
}
```

Uniform function and parent module

- Upon sending a message to the node, the hub selects a random node m.
- We sample the node index from a *Uniform* distribution .

```
Int m = int(uniform(0,n))
```

- How to get n ??? Remember that n is a parameter defined in the **network** (the compound system module).

```
getParentModule()->par("n")
```

- Call this inside Hub.c. It returns a pointer to the parent module Network, and then we can use this pointer to access any parameter in the parent module

Sending a packet through a gate array

- Finally, we can use the send(msg, gatename, gateindex) function to send our message on the proper gate to our destination node.

Send(msg, “outs”, 1);

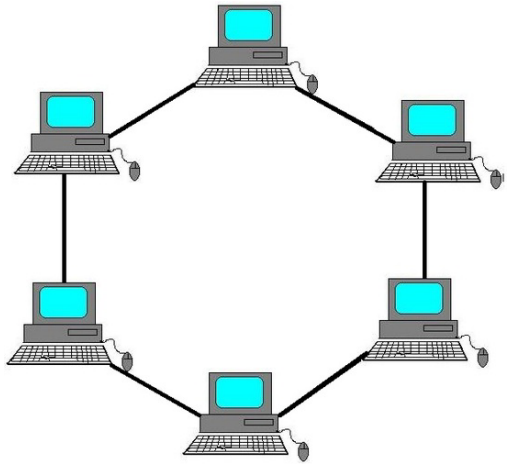
Useful functions

- **gateSize("outs")** //inside c, returns the size of the "out" gate array
- **getIndex()** // inside c, returns the current node index
- **getParentModule()->par("n")** //inside node.c, returns a pointer to the parent module Mesh, and then we can use this pointer to access any parameter in the parent module.
- **Network.n** // inside .ini where Network is the system module.
- **int=getKind(), setKind(int)** In the cMessage class, there is a data member called "Kind" a short int , can be used as an identifier for the message. The "Kind" has setter and getter functions just like the "Name" data member.

Today's lab requirement number 2/2:

- Create a ring network topology consisting of N nodes.
- N should be configurable in the .ini file.
- Before starting, every node prints to the console a message " Starting Node i", where i is the node index.
- In a way to imitate what happens in real world dynamic networks, we want to enable Node 0 to explore the topology and find the number of the connected nodes in the network through messages exchanging. [without the use of getParent() for the parameter N here].
- After a complete exploration phase , Node 0 , should print to the console the number of nodes in the network N and finish the simulation.

Ring



Submission

- Work in pairs
- Delivery in two weeks
- Submit two .zip project folders for each problem starts with p1_ or p2_ then your names.
- In the private comments write your names and Id's.
- We will need a discussion for this lab.