# Report for CN-P2P Project

## Definitions

**Network Nodes:**

('127.0.0.1',20000),

('127.0.0.1',20001),

('127.0.0.1',20002),

('127.0.0.1',20003),

('127.0.0.1',20004),

('127.0.0.1',20005)

**Note**: Node ID's are defined as port-2000

**Unidirectional neighbour:**

Node A becomes unidirectional neighbour of B if A sends B a packet and

1)B is not in A's unidirectional or bidirectional neighbours list

or

2) if B is inside either of the lists but A has not sent a hello packet to B within 8 seconds of the last time he sent a packet to him.

**Bidirectional neighbour:**

node A becomes bidirectional neighbour of B if

1)B has less than 3 bidirectional neighbours at that moment

and

2)A sends a hello packet to B within 8 seconds after sending the last packet to B

and

3)B is listed in either unidirectional or bidirectional neighbours of A.

# Explanations

**How does finding neighbours work?**

Each node checks his list of bidirectional and unidirectional neighbours every 2 seconds. If the number of its bidirectional and unidirectional neighbours is less than N (Why do we include unidirectional neighbours too? Because we can send them a hello every 2 seconds and there is a potential of becoming bidirectional neighbours.), he picks a random node (specified by its IP address and Port ) from the list of potential neighbours (gathered by excluding his own address and his unidirectional and bidirectional neighbour's address from the list of all network node addresses) and sends a Hello packet to that node. He also sets his "tempN" field to this node's address.

Each time a packet is received our node checks if "tempN" has a non-null value and this packet is from "tempN".If so he sets "tempN" to null. And treats the packet like any other packet received.

Then he checks if the packet was sent to him within the 8 second threshold by checking "lastRecFrom". If it was then he checks if his ID is inside the node's unidirectional or bidirectional neighbours.
If it is then he adds this node to his bidirectional neighbours (he won't do anything if this node is already his bidirectional neighbour).
Otherwise the node is added to his unidirectional neighbours.

On the other hand if the packet wasn't sent within threshold he treats the sender as a unidirectional neighbour.

**Node Reachability:**

We defined reachability of each node A for another node B as the total amount of time A has been inside B's unidirectional or bidirectional neighbours.

**Network Topology:**

To draw the network topology from a node's point of view, we looked at the history of that node's neighbours- more specifically the last neighbours list each of its neighbours has sent.

This list contains both a list of unidirectional and bidirectional neighbours of our node's neighbour.

We only use the bidirectional neighbours history of our neighbours to draw this topology.

<u>There is an edge from node A to node B if B is inside A's bidirectional neighbours.</u>

**Packet loss:**

To simulate 5% chance of packet loss:

before receiving each packet a node picks a random number between 0 and 99. If this number is less than 5 he doesn't process that packet.

**How are the network nodes started, paused and stopped?**

We used multiprocessing to create K processes each running code for a single node. An instance of JoinableQueue (pause_queue) was passed to each node for synchronization purposes.

Every 20 seconds a random node has to be paused for 10 seconds. To do that our network selects a random node ID and puts this ID inside pause_queue for every node inside network.

Each node checks pause_queue every 20 seconds. If the ID inside pause_queue is equal to its own ID, he has to sleep for 10 seconds. If ID is equal to “None” he has to output results, stop execution and exit.

The network has to keep track of which nodes are currently awake and which ones are sleeping. So it keeps a list called “awake_nodes” storing the ID’s of the nodes not asleep. Every 10 seconds network wakes up, updates “awake_nodes” and randomly chooses a node from it. Another list called “to_be_woken” and a binary variable “turn” is kept to keep track of which node is supposed to be added to “awake_nodes”.

**Note**:

Receiving packets is done in a non-blocking manner so that our node can periodically check for pause/abort signal from network, find new neighbour and send hello to neighbours .

Here’s a piece of log data after running network (See the interations between node 4 and 5 in the highlighted lines):

```
[21:15:20.293387]-4: node started
[21:15:20.293744]-4:possible nodes are [0, 1, 2, 3, 5]
[21:15:20.293788]-4:randomly sending neighbour req to ('127.0.0.1', 20005)
[21:15:20.293815]-4:sending hello to 127.0.0.1:20005
[21:15:20.297686]-5:received hello from 4
[21:15:20.297736]-5:node 4's neighbours are:{'unidir': [], 'bidir': []}
[21:15:20.297758]-5:my neighbours are: unidir:[] bidir:[]
[21:15:20.297782]-5:node 4 now my unidir neighbour
[21:15:20.309514]-1:received hello from 3
[21:15:20.309566]-1:node 3's neighbours are:{'unidir': [], 'bidir': []}
[21:15:20.309585]-1:my neighbours are: unidir:[] bidir:[]
[21:15:20.309608]-1:node 3 now my unidir neighbour
[21:15:22.274216]-1:sending hello to 127.0.0.1:20003
[21:15:22.277975]-3:received hello from 1
```

[21:15:22.278023]-3:node 1's neighbours are:{'unidir': [3], 'bidir': []}
[21:15:22.278046]-3:my neighbours are: unidir:[] bidir:[]
[21:15:22.278069]-3:node 1 now my bidir neighbour
[21:15:22.279907]-5:sending hello to 127.0.0.1:20004
[21:15:22.288945]-3:sending hello to 127.0.0.1:20001
[21:15:22.289420]-3:possible nodes are [0, 1, 2, 4, 5]
[21:15:22.289449]-3:randomly sending neighbour req to ('127.0.0.1', 20005)
[21:15:22.289469]-3:sending hello to 127.0.0.1:20005
[21:15:22.289726]-5:received hello from 3
[21:15:22.289762]-5:node 3's neighbours are:{'unidir': [], 'bidir': [1]}
[21:15:22.289781]-5:my neighbours are: unidir:[4] bidir:[]
[21:15:22.289801]-5:node 3 now my unidir neighbour
[21:15:22.289851]-5:possible nodes are [0, 1, 2, 3, 4]
[21:15:22.289871]-5:randomly sending neighbour req to ('127.0.0.1', 20000)
[21:15:22.289890]-5:sending hello to 127.0.0.1:20000
[21:15:22.290569]-1:received hello from 3
[21:15:22.290612]-1:node 3's neighbours are:{'unidir': [], 'bidir': [1]}
[21:15:22.290631]-1:my neighbours are: unidir:[3] bidir:[]
[21:15:22.290653]-1:node 3 now my bidir neighbour
[21:15:22.293835]-4:received hello from 5
[21:15:22.293878]-4:node 5's neighbours are:{'unidir': [4], 'bidir': []}
[21:15:22.293899]-4:my neighbours are: unidir:[] bidir:[]
[21:15:22.293922]-4:node 5 now my bidir neighbour

# Results

All results are store in **output** folder.

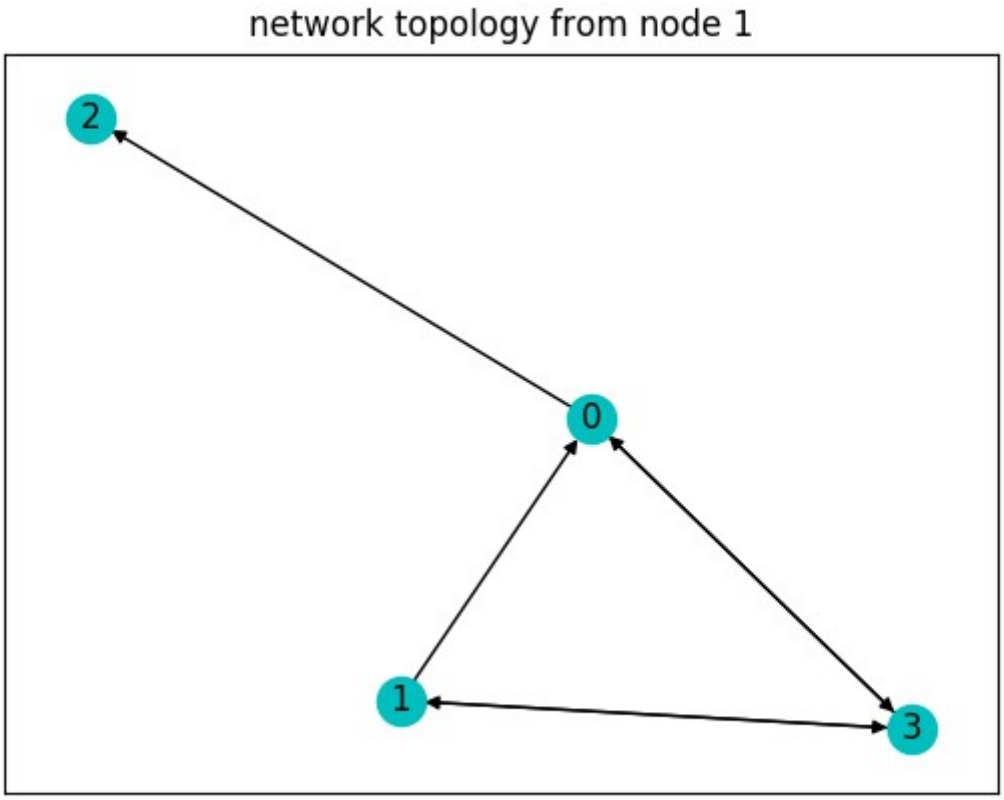
**for each node N:**

1) **packets**: List of all nodes N has ever connected to with their ip and port and packets received from/sent to them
2) **neighbours**:List of bidirectional neighbours with a maximum count of 3
3) **reachability:** Reachability of N's neighbours
4) **connections:** topology of network at the end from N's point of view

Below is a final output of **node 1** inside **1_history.json.**


```
{
    "addresses": {
        "0": "('127.0.0.1', 20000)",
        "1": "('127.0.0.1', 20001)",
        "2": "('127.0.0.1', 20002)",
        "3": "('127.0.0.1', 20003)",
        "4": "('127.0.0.1', 20004)",
        "5": "('127.0.0.1', 20005)"
    },
    "reachability": {
        "0": "0:00:02.010235",
        "2": "0:00:42.009683",
        "3": "0:00:02.003876",
        "5": "0:00:09.999292"
    },
    "packets": {
        "0": {
            "sent": 2,
            "received": 1
        },
        "2": {
            "sent": 18,
            "received": 9
        },
        "3": {
            "sent": 15,
            "received": 11
        },
        "4": {
            "sent": 0,
            "received": 0
        },
        "5": {
            "sent": 5,
            "received": 0
        }
    },
    "neighbours": [0,3],
    "connections": {
        "0": {
            "bidir": [2,3],
            "unidir": [1]
```

```
        },
        "1": {
            "bidir": [0,3],
            "unidir": []
        },
        "3": {
            "bidir": [0,1],
            "unidir": []
        }
    }
}
```


*1_topology.png*