COP5615 – FALL 2019

PROJECT 2 – GOSSIP SIMULATOR

(September 10, 2019 – October 01, 2019)

GROUP MEMBERS

- Arpan Banerjee (UFID 9359-9083)
- Krutantak Patil (UFID 5615-6343)

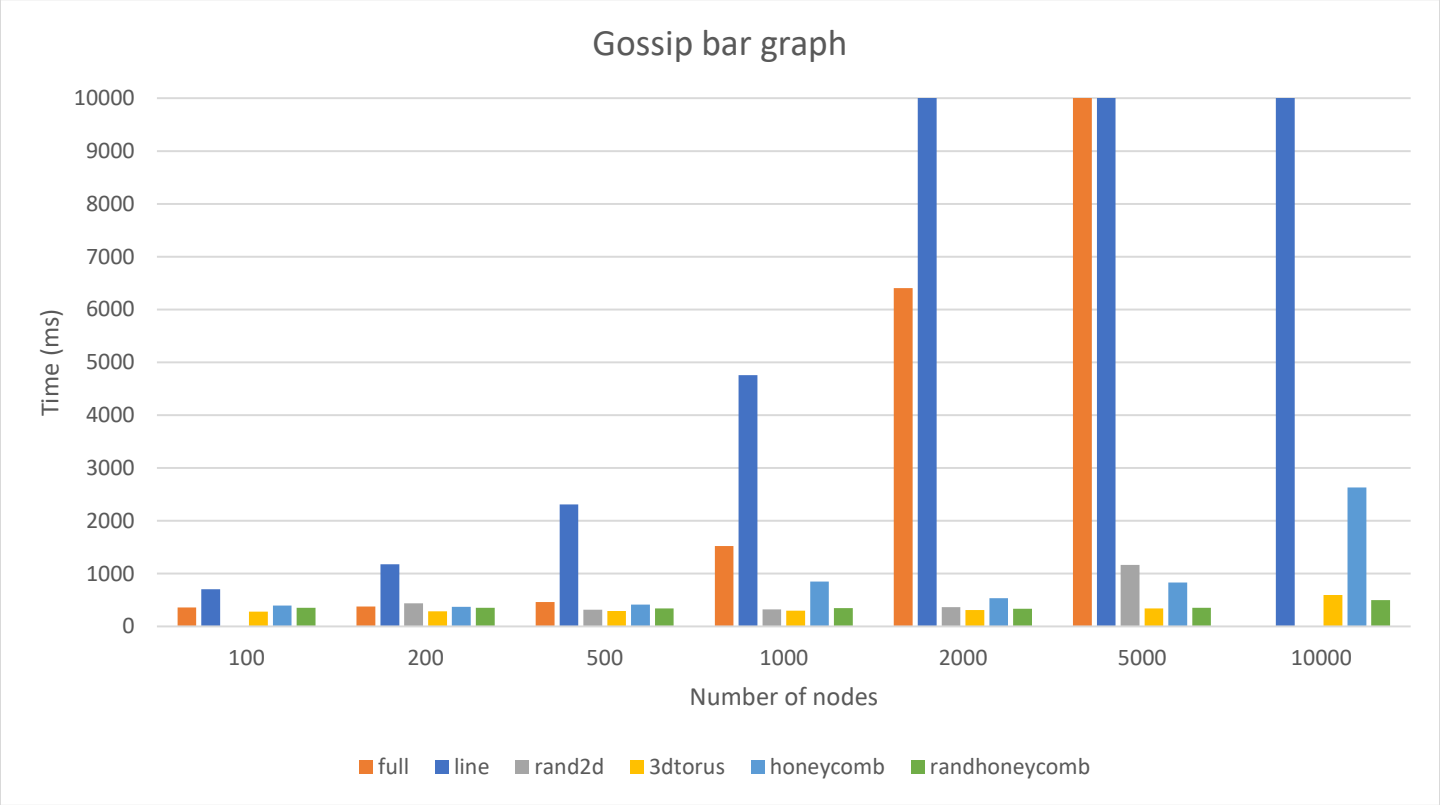
TOPOLOGIES

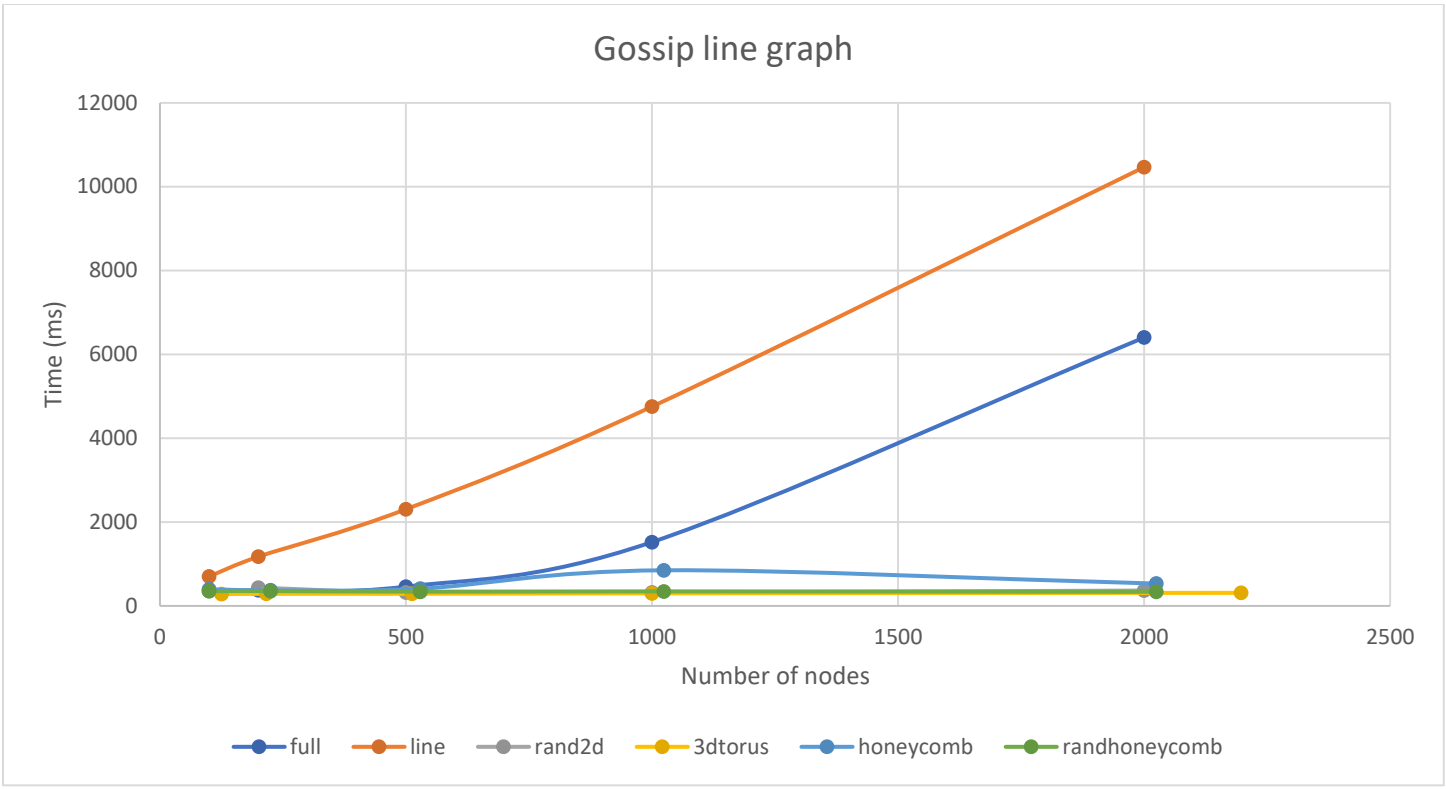
- Full Network (full) – Process has all the other processes as its neighbors. With **n** processes in the network, all processes will have **n-1** neighbor processes. Running full network for large network (above 5000) takes a long time to create the topology (generate the neighbor lists).
- Line (line) – All processes are placed in a linear order forming a line, where a process at **i^{th}** position has neighbors at positions **i-1** and **i+1**. 1st and the last process will have only **1** adjacent neighbor, whereas, all the other process will have **2** neighbor processes.
- Random 2D Grid (rand2D) – All processes are randomly positioned at x, y coordinates on a [0-1.0] x [0-1.0] square. Two processes are connected if they are within **.1** distance to other process. Processes in this topology will have random number of neighbor process.
  Rand2D does not work well when the number of nodes is small because the nodes are sparsely placed, and some nodes may not have any neighbors. In this case, the program might get stuck before reaching convergence.
- 3D Torus Grid (3Dtorus) – Processes are placed in a 3-dimensional cube and each process will have **6** neighbor processes. Processes on the outer surface are connected to other actors on opposite side.
- Honeycomb (honeycomb) – Processes are arranged in form of hexagons and two processes are connected if they are connected to each other. Each process will have maximum of **3** neighbor processes.
- Honeycomb with a random neighbor (randhoneycomb) – Process are arranged in form of hexagons. Neighbor processes are derived using Honeycomb Topology and an extra neighbor process is randomly chosen from the process in the network. Each process will have maximum of 4 neighbor processes.

GOSSIP PROTOCOL

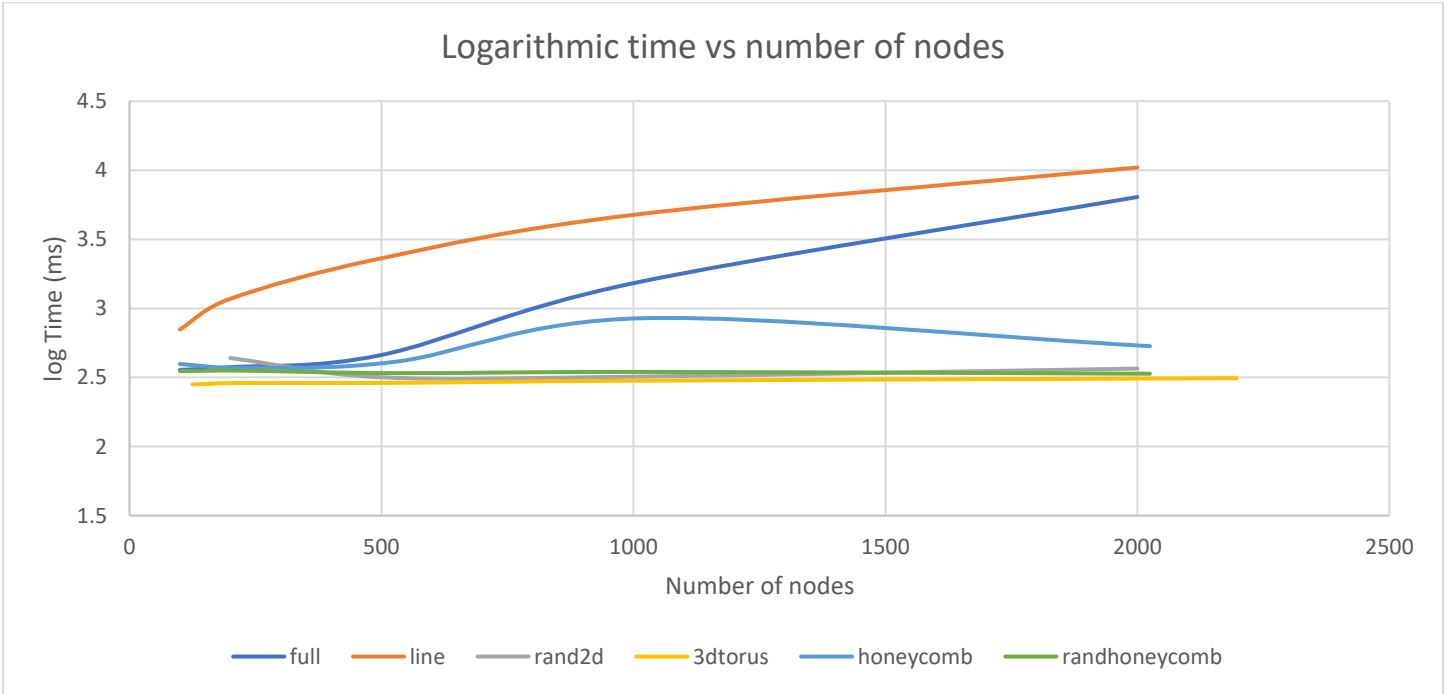
- Implementation –
  - Each node has the following values in a tuple as its state –

        - **Rumor** – Message passed between processes in the topology
        - **Rumor Count** – Maximum count for each process is set to **20**. Once the process reaches maximum rumor count, it stops transmitting
        - **Stats PID** – Process id of a separate timekeeping process for maintaining start and end timer
        - **Failure Probability** – Required for failure model
    - Initially the main process will select a gossip process at random and transmit a rumor **"psst"** to the process.
    - Once a process receives a rumor, it will increment its rumor count by 1, i.e. rumorCount = rumorCount + 1.
    - Each process, after receiving a rumor for the first time, will start transmission loop which sends a message to its neighbor process, selected at random, every 10 milli seconds.
    - A process stops transmitting rumors to its neighbor processes, once the rumor count reaches maximum limit.
    - Timekeeping process reports back the total convergence time to the main process, once all the processes have reached maximum rumor count or pre-defined number of processes have reached maximum rumor count.
- <u>Observations</u> –
    - Below graph shows the convergence time for different topologies and number of nodes. Note that we have averaged the times over 5 runs.

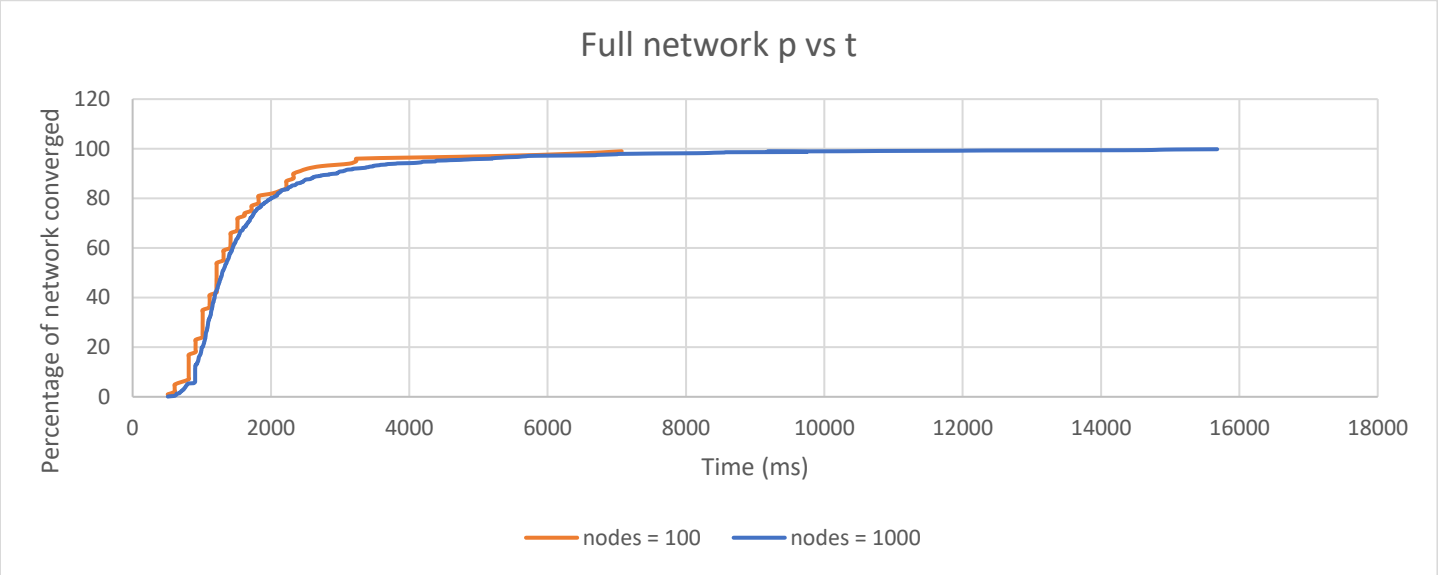




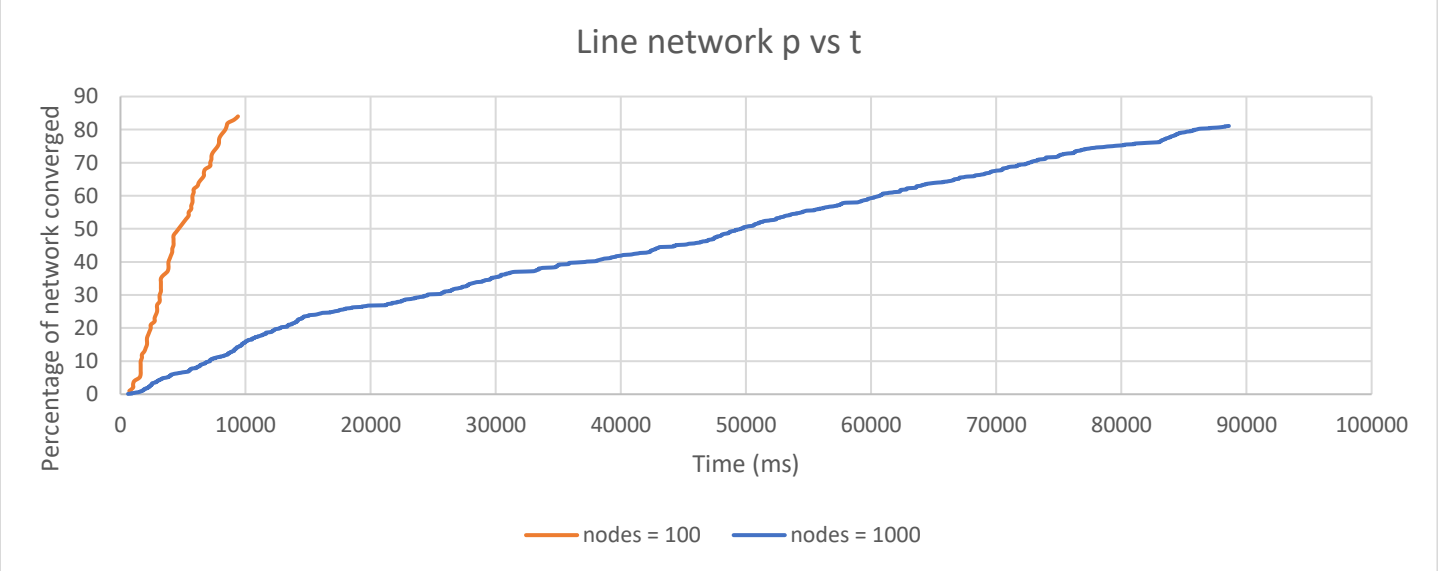
- Good topologies which scale well for gossip protocol are rand2D, 3DTorus, Honeycomb and random honeycomb (which have a constant number of neighbors) for large number of nodes.
- The convergence time for line topology increases almost linearly as we increase the number of nodes which is expected.
- Below is a graph of logarithmic time vs number of nodes.



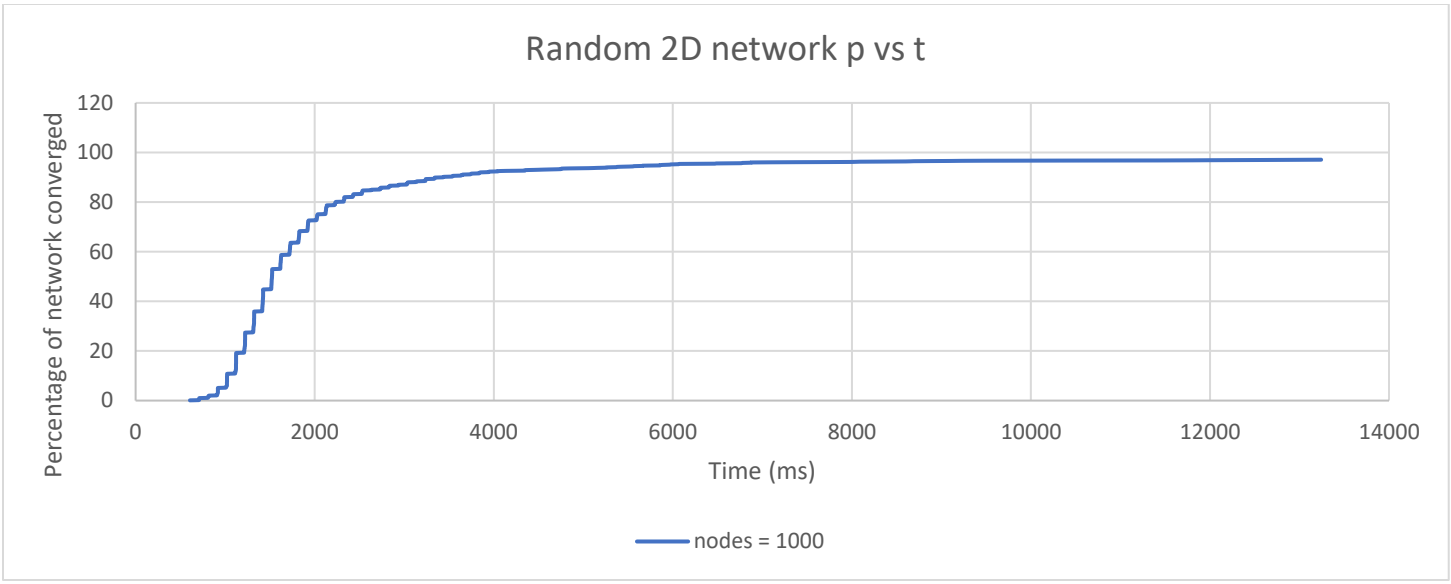
- We have also plotted percentage of network converged vs time graphs for each network which allows us to see at what rate the gossip spreads through the network.



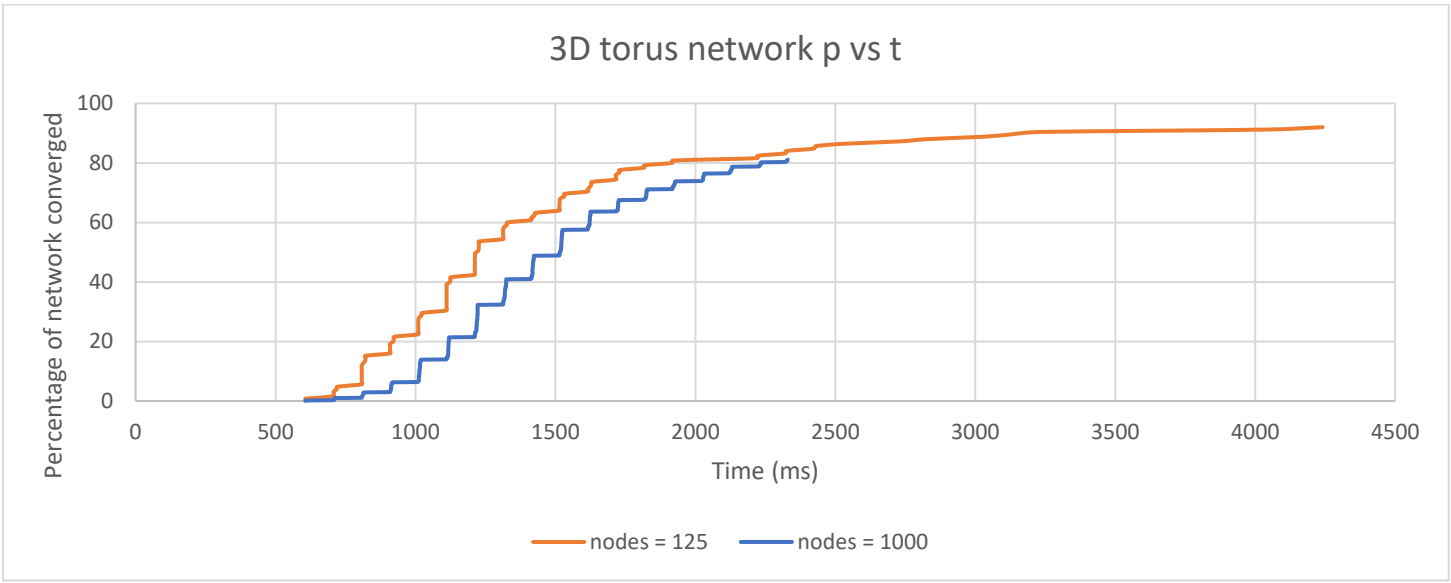
- We see that the spread starts slow but once a few nodes get the message, there is an exponential spread since all the nodes are connected and it tapers off towards the end when a lot of the nodes shut down.



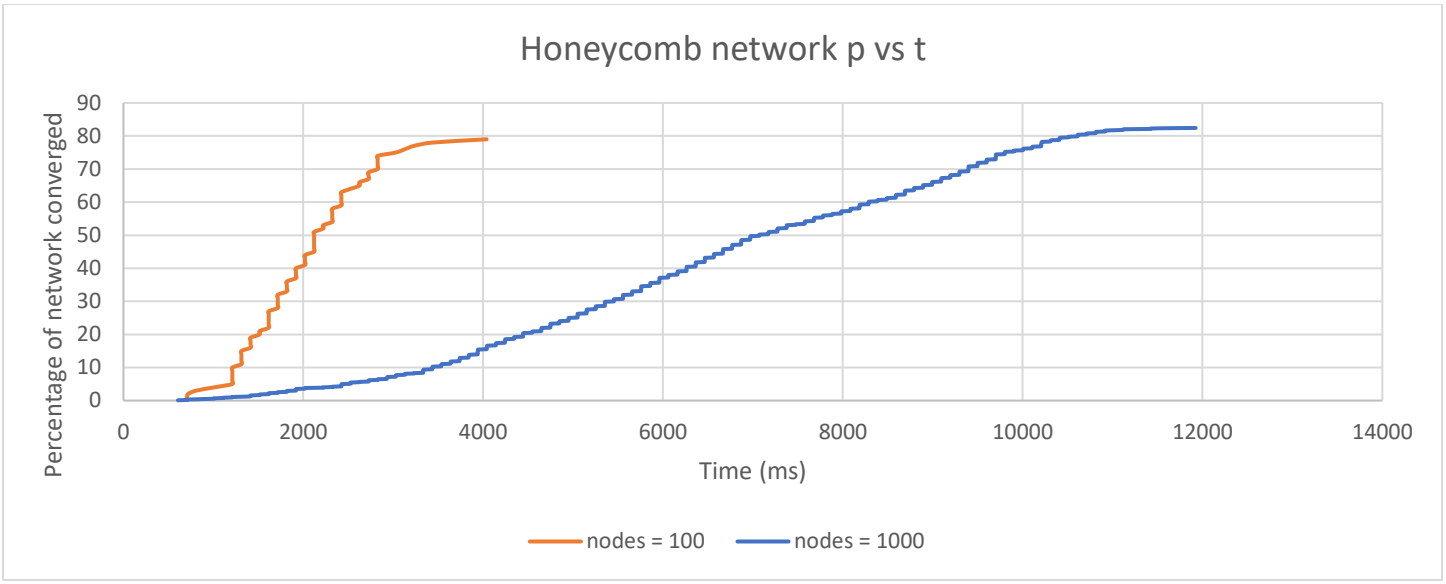
- In a line network the spread is linear since each node is only connected to one node forward and one node backward, so it can spread in at best linear time.



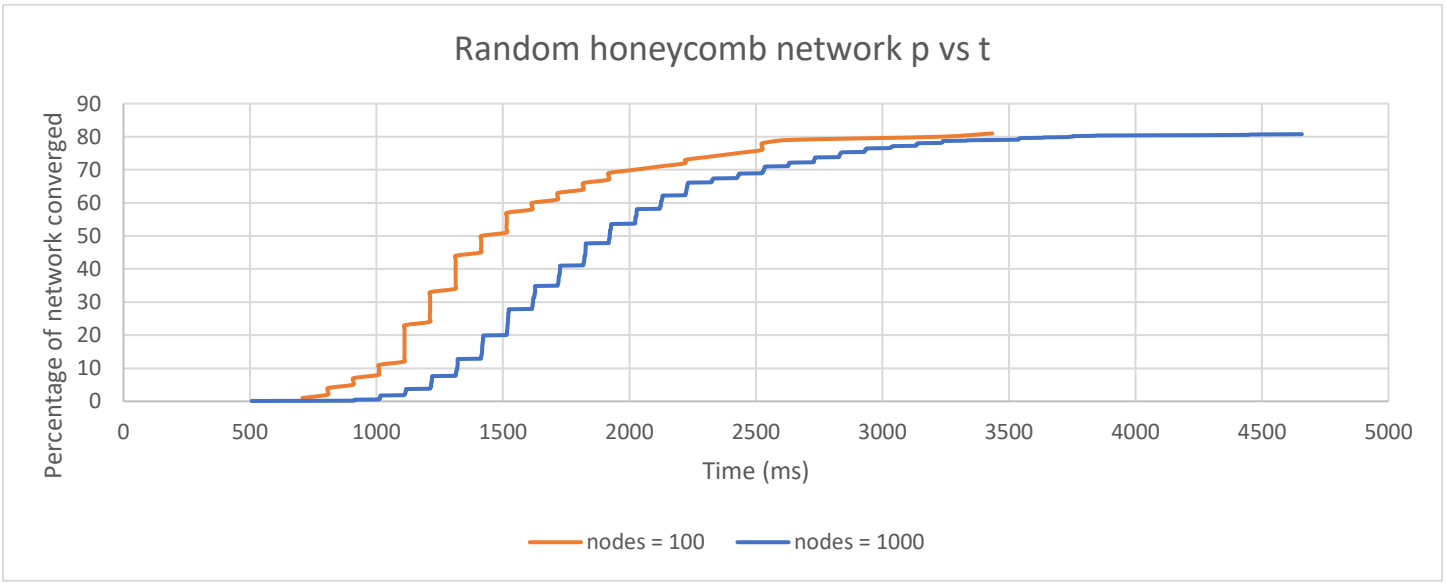
- Random 2D network is like many localized full networks when the number of nodes is large hence it has a spread pattern similar to the full network.



- 3D torus network is in between a linear spread and an exponential spread like the full network. That is because it has properties of both types of networks – it has a constant number of neighbors (6) like the linear case, but it is also connected to the opposite sides at the edges which spreads the gossip much farther.



- A honeycomb network is also like a line network where every node has 3 neighbors instead of 2 hence the spread pattern is almost linear.

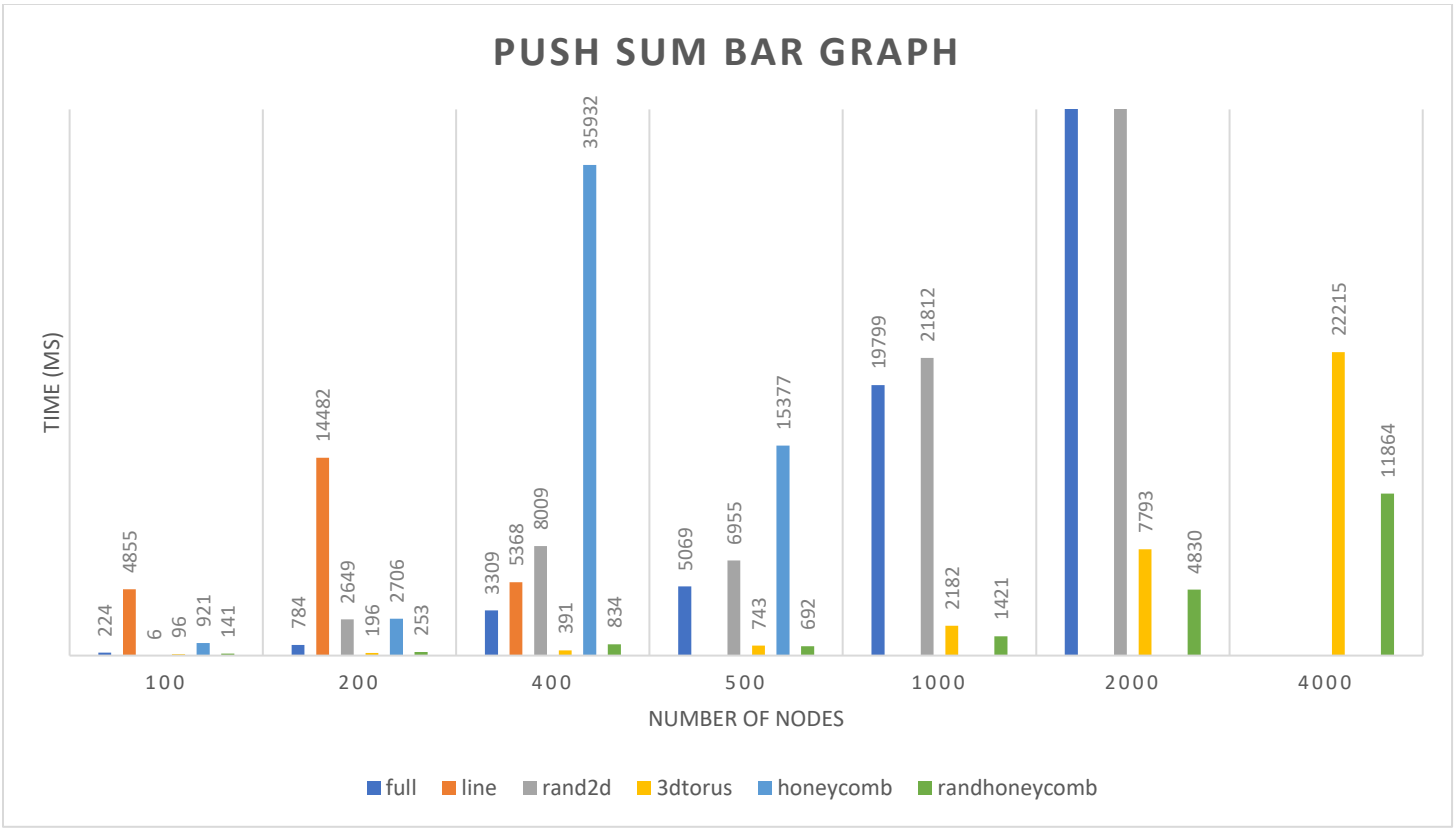


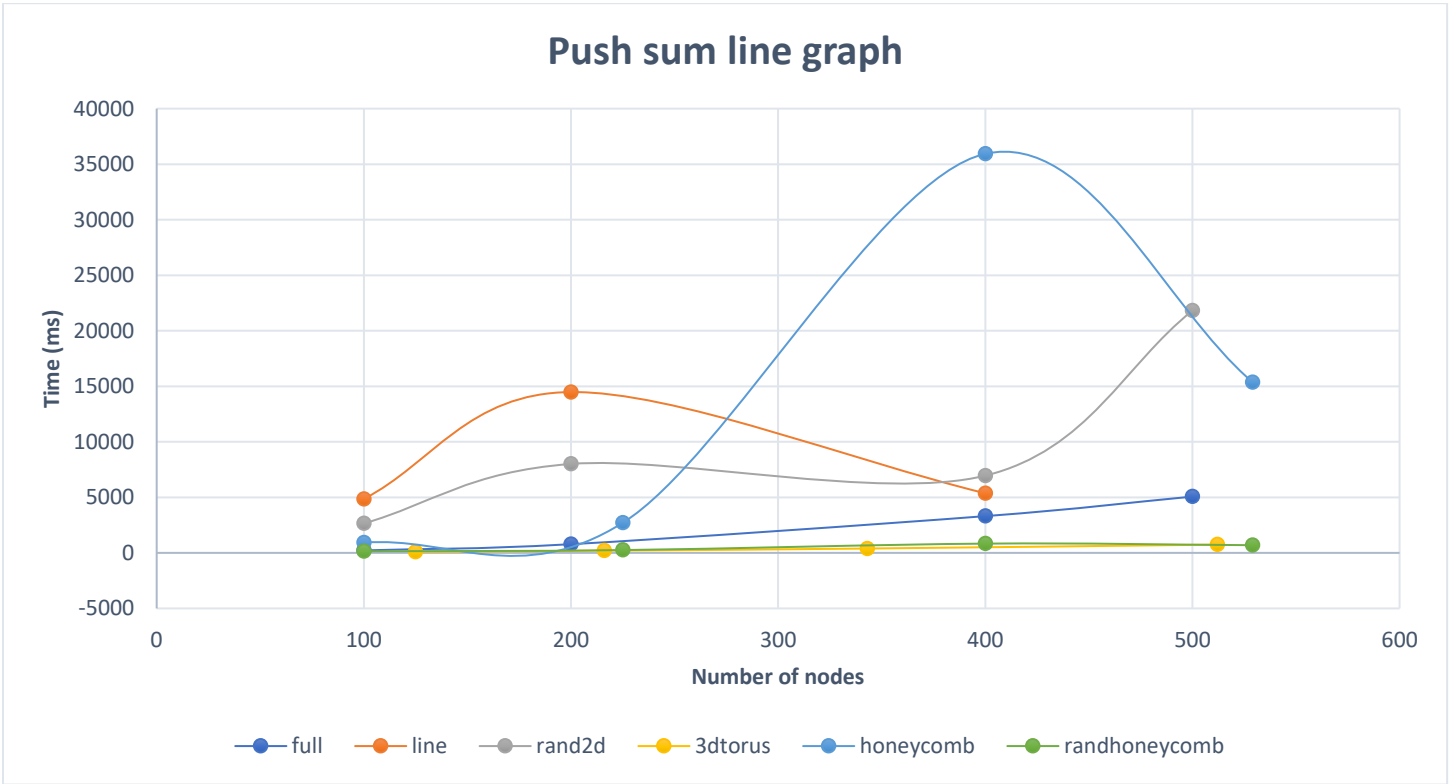
- A random honeycomb network is simply a honeycomb network with one additional random neighbor per node. This helps the gossip to spread faster as it removes the localized barrier of a regular honeycomb network.

PUSH-SUM PROTOCOL

- Implementation –
  - Each node has following values in a tuple as its state –
    - **Neighbors** – List of process ids of its neighbors decided as per the topology
    - **Sum** – It is initialized to index value of the process

        - **Weight** – It is initialized to 1
        - **Ratio** – It is the sum estimate which is (sum / weight)
        - **Round** – To keep track of number of rounds
        - **Stats PID** – Process id of a separate timekeeping process for maintaining start and end timer
        - **Running** – Flag to determine whether a process can transmit a message or has reached termination condition and hence cannot transmit or receive messages over the network
        - **Failure Probability** – Required for failure model.
    - Initially the main process will select a push-sum process at random and transmit a message with sum=0 and weight=0 to that process.
    - Once a process receives a message with sum=s and weight=w, it will update its sum and weight i.e. sum=state.sum + s and weight=state.weight + w.
    - A process will select a random process from its neighbor list and then transmit message, with sum=new_state.sum/2 and weight = new_state.weight/2, if the selected process is running.
    - A process terminates, i.e. state.running is updated to false, if its ratio i.e. sum estimate does not change more than 10^{-10} in 3 rounds
    - A process with no neighbors in its list of neighbors terminates and notifies the timekeeping process to terminate all running processes in the network topology (since there is only one process in the network at one time, the protocol cannot proceed further).
    - Timekeeping process will report the converge time for the network once every node in the network has converged.
- Observations –
    - The graph below shows the convergence or termination time taken by various topologies for different number of nodes in a Push sum protocol.





- Push sum is a sort of linear propagation where there is only one message in the network at a time. Thus, we can once again see that topologies with a constant number of neighbors perform well, namely, 3d torus, honeycomb, random honeycomb.

- However, in push sum, sometimes the network may get prematurely terminated when a node's neighbor list becomes empty.
- Below is a graph of logarithmic time vs number of nodes.