Assignment 2: Jamming Attack Simulation in ns-2

- Pramod George Jose (Roll 13)

An overview of how to execute the code and how it works:-

You should find 5 code files along with this writeup- run_ns_mp.py, wireless_jam.tcl, scen_wireless_jam.tcl, conn_wireless_jam.tcl and events_wireless_jam.tcl.

To run the program, you need python3 and ns2.35 (and xgraph if you want to view the graphs). Type the command in a terminal:-

```
python3 run_ns_mp.py
```

The TCL script consists of 4 code files: wireless_jam.tcl, scen_wireless_jam.tcl, conn_wireless_jam.tcl and events_wireless_jam.tcl. The first file, wireless_jam.tcl, is the main file and calls the procedures and utilises the code written in the other 3 files. The scen_wireless_jam.tcl file creates new nodes and places them at random co-ordinates. The conn_wireless_jam.tcl file defines the connections among the nodes. For this project, the benign nodes only connect with other benign nodes; whereas malicious jammer nodes connect with every node (both benign and malicious). The events_wireless_jam.tcl file defines the events that should occur during the simulation – things like when the malicious nodes should start and stop jamming. In this way, the code is modularized and this organization makes it easy to debug and modify the code without breaking other parts of the code.

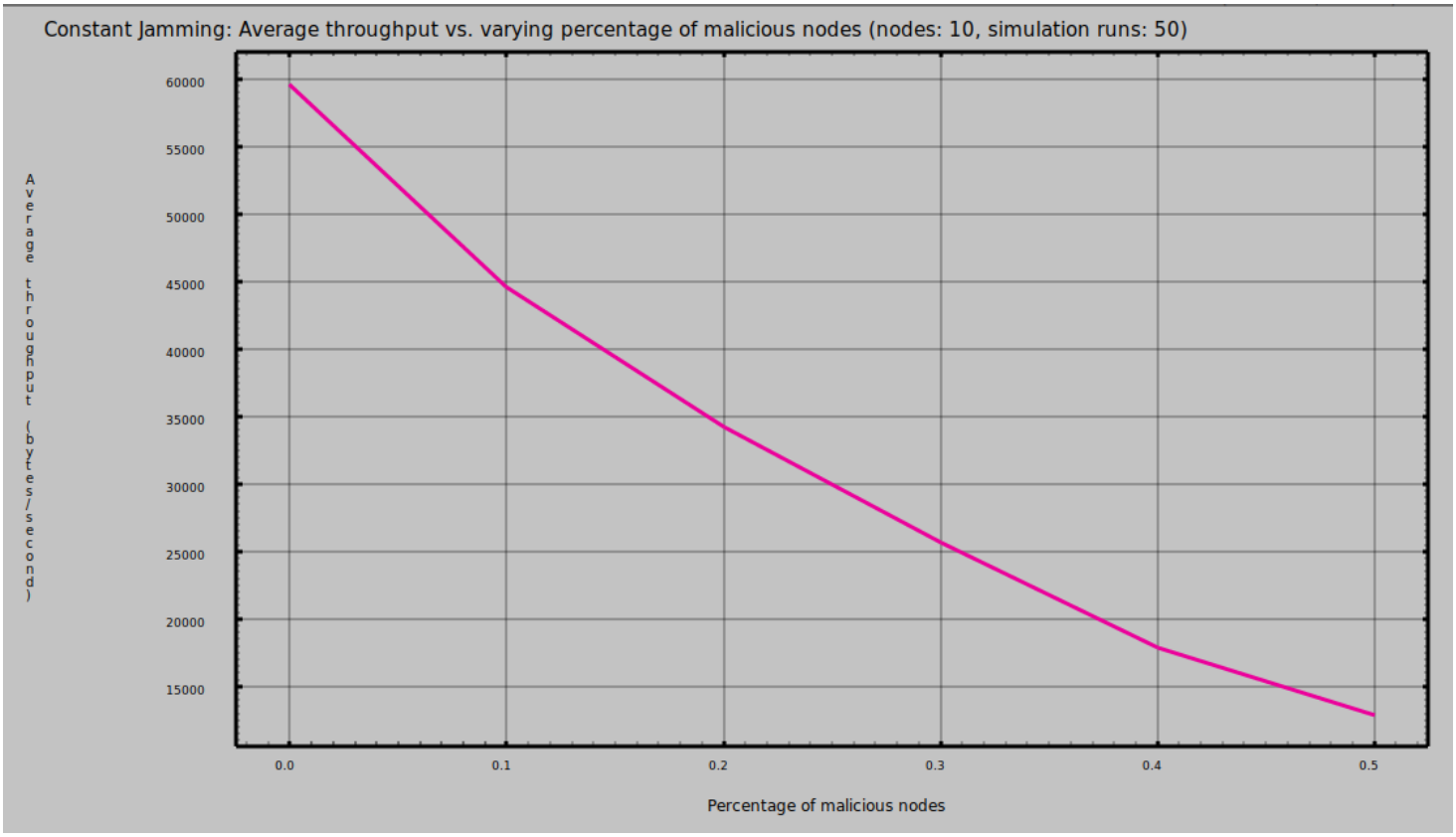
The wireless_jam.tcl script takes 6 arguments, the first two being the total number of nodes and the percentage of malicious nodes in the adhoc network respectively. The wireless_jam.tcl script creates a network scenario based on the input arguments. The connections among the benign nodes and the locations of the nodes are random; thereby simulating a real-life wireless situation. Therefore, the purpose of the TCL script is only to create a particular wireless scenario.

The purpose of the Python script run_ns_mp.py is to execute the TCL script with varying argument values and assimilate the results from the TCL script into 6 graphs. The Python script executes multiple instances of the TCL script in parallel to speed up the simulation. It also executes the TCL script 50 times and takes an average of the results thus making the final results more accurate. In short, it is responsible for trying each of the 3 attacks, running each attack 50 times, analysing the trace file for each run and calculating a running average which is finally written to 6 graph files.

The following are the same for all simulations carried out in this report:

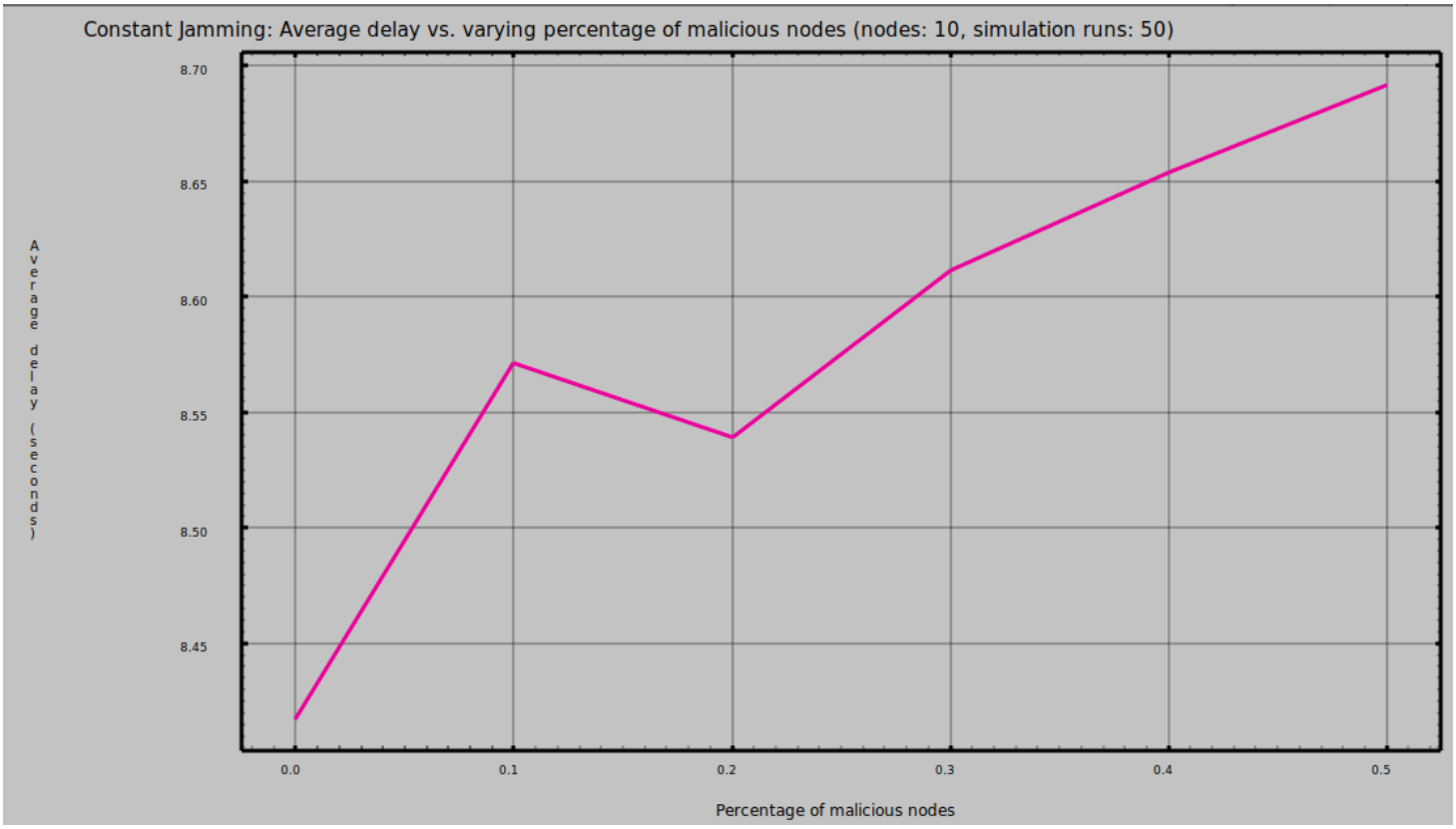
1) There are a total of 10 nodes, out of which some percentage (0.0 to 0.5; both inclusive) of the nodes would be made malicious. The number of nodes and the percentage of malicious nodes are sent as arguments to the TCL script.
2) Configurations for a legitimate sender: CBR on top of UDP. The CBR agent sends packets at a rate of 200,000 (default is 448,000).
3) Configurations for a legitimate receiver: A LossMonitor Agent with default configuration parameters.
4) Configurations for a legitimate sender: CBR on top of UDP. The CBR agent sends packets of size 8192 bytes (default is 210 bytes) at a rate of 5,000,000 (default is 448,000).
5) Configurations for a legitimate receiver: A Null Agent with default configuration parameters.
6) The DSDV routing protocol is used.
7) All nodes are activated at the 0^{th} second.
8) Each simulation runs for 20 seconds.
9) The scene dimensions are 175 x 175.

Constant Jamming:



Constant Jamming: Average throughput vs. varying percentage of malicious nodes (nodes: 10, simulation runs: 50)

As we can see from the throughput graph, it decreases as the number of malicious nodes increase. There is a linear decrease from 60,000 bytes per second (60 Kbps) to 15,000 (15 Kbps).



Constant Jamming: Average delay vs. varying percentage of malicious nodes (nodes: 10, simulation runs: 50)

As we can see from the graph above, the delay is growing almost linearly with respect to the percentage of malicious nodes. It increases from an average of 8.4 seconds to roughly 8.7 seconds.

Therefore, it is clear that with the increase in the percentage of malicious constant jammer nodes in the network, the performance decreases – the throughput decreases and the delay increases.

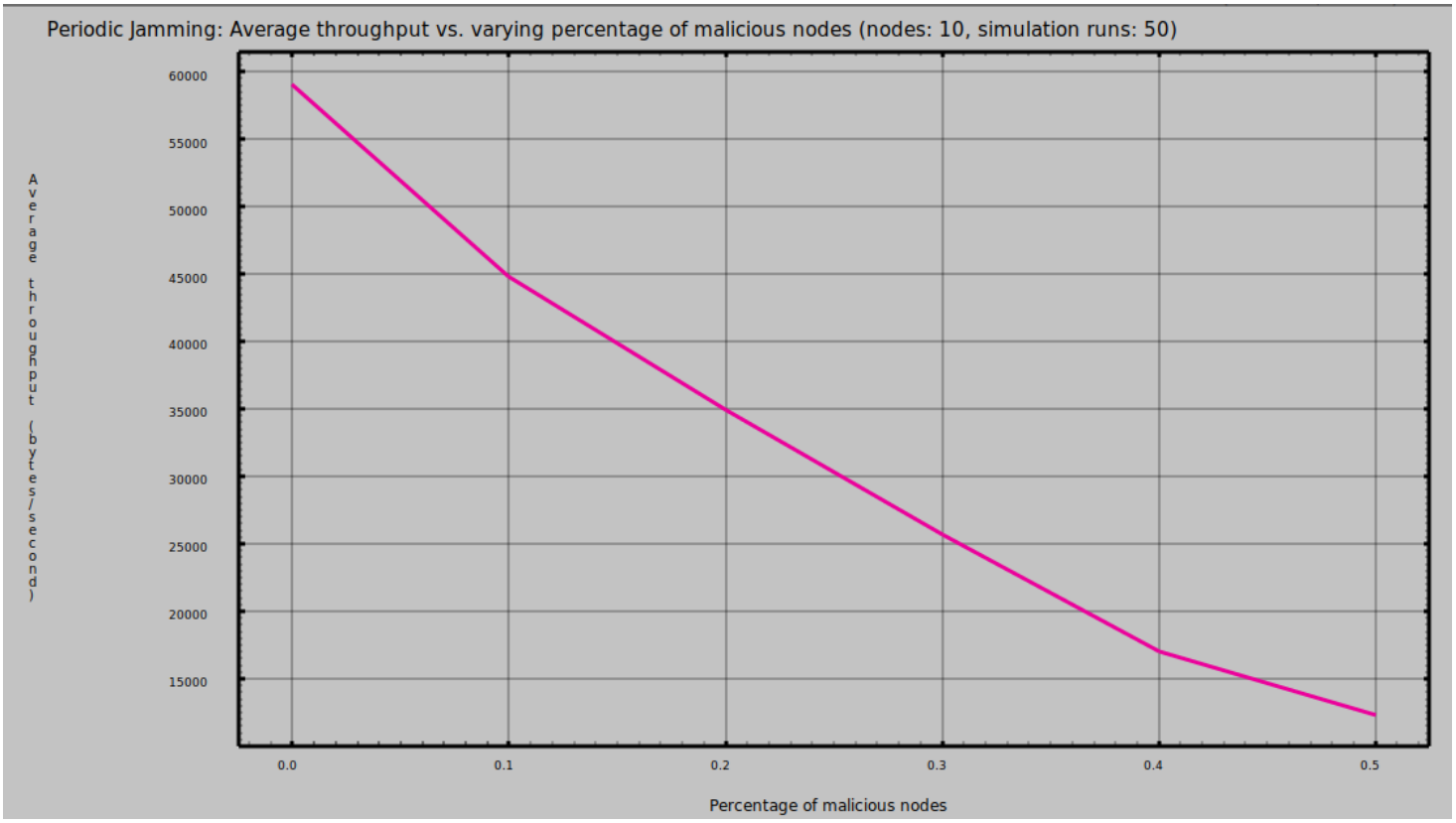
Periodic Jamming:

The jammer nodes have random periods chosen from the range [2, 5]. For example, if a jammer has the period as 3 seconds, then it would be active from 0-3 seconds, then inactive from 3-6 seconds, then active from 6-9 seconds, and so on and so forth until the completion of the simulation.

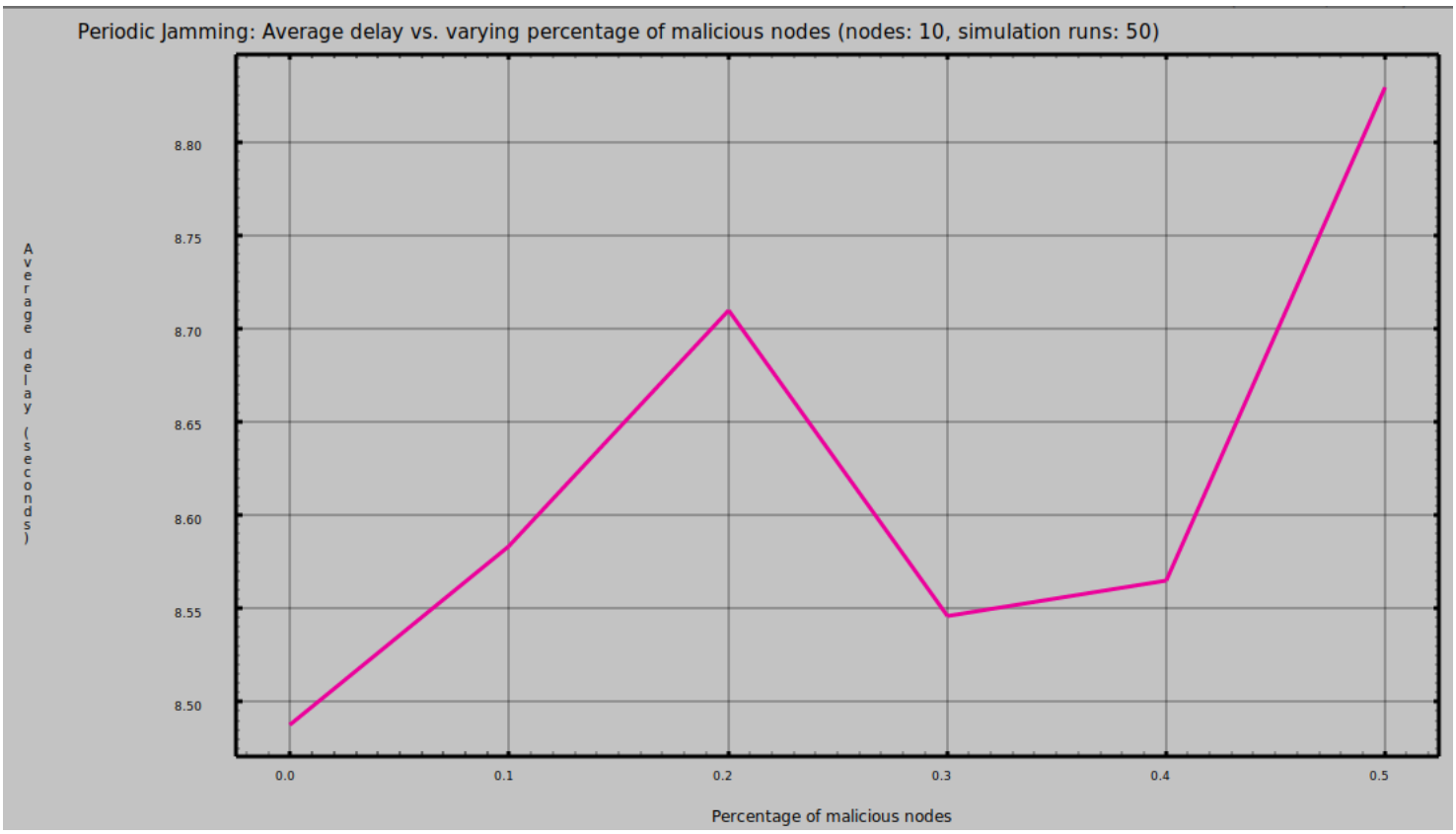
From the throughput graph, it is clear that the throughput decreases linearly with the increase in the number of malicious nodes. It decreases from 59,000 bytes per second (59 Kbps) to 12,356 bytes per second (12.356 Kbps).

The delay clearly increases as the number of jammer nodes increase. The average delay increases from 8.48 seconds to 8.82 seconds.

Therefore, it is clear that with the increase in the percentage of malicious periodic jammer nodes in the network, the performance decreases – the throughput decreases and the delay increases.

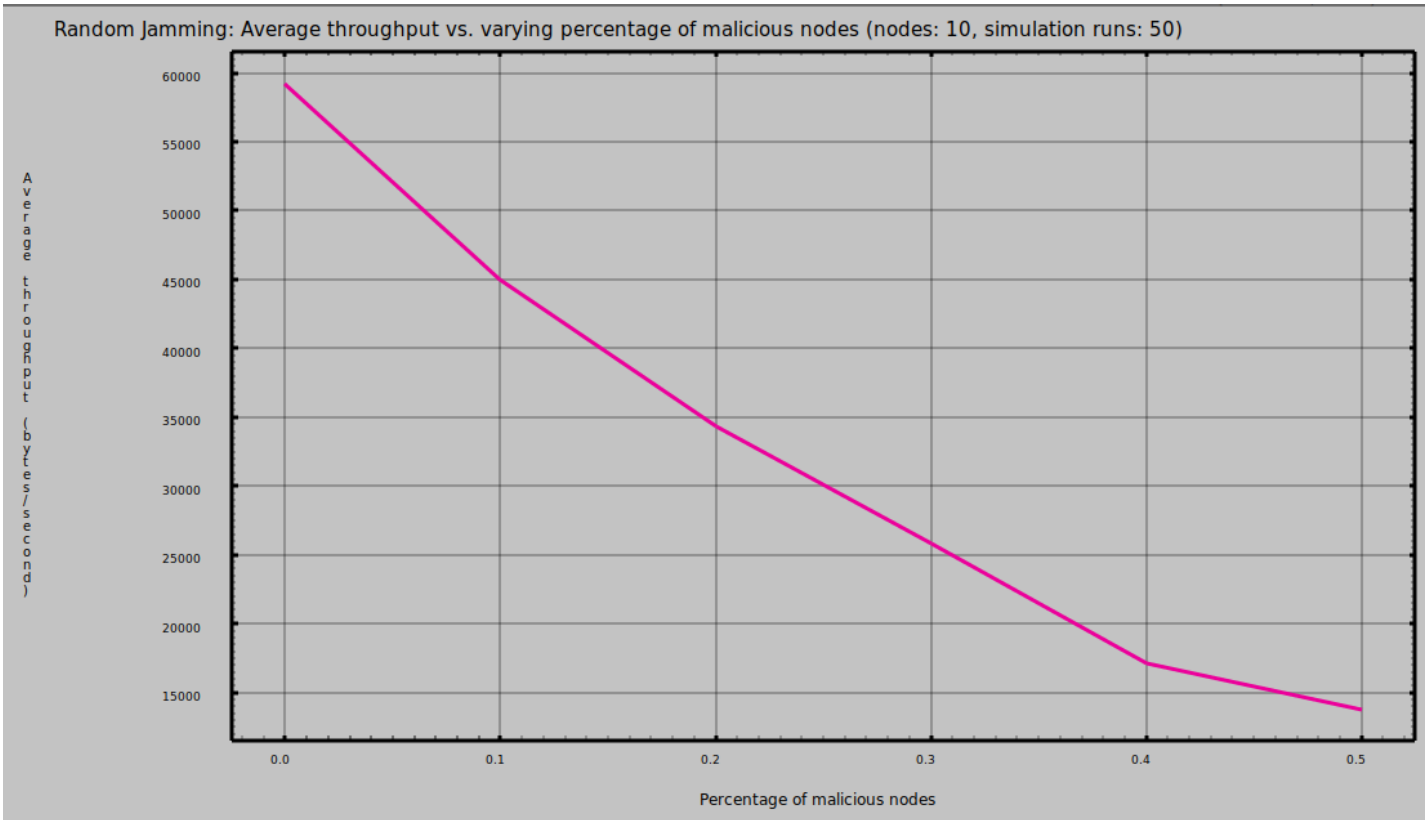


Periodic Jamming: Average throughput vs. varying percentage of malicious nodes (nodes: 10, simulation runs: 50)

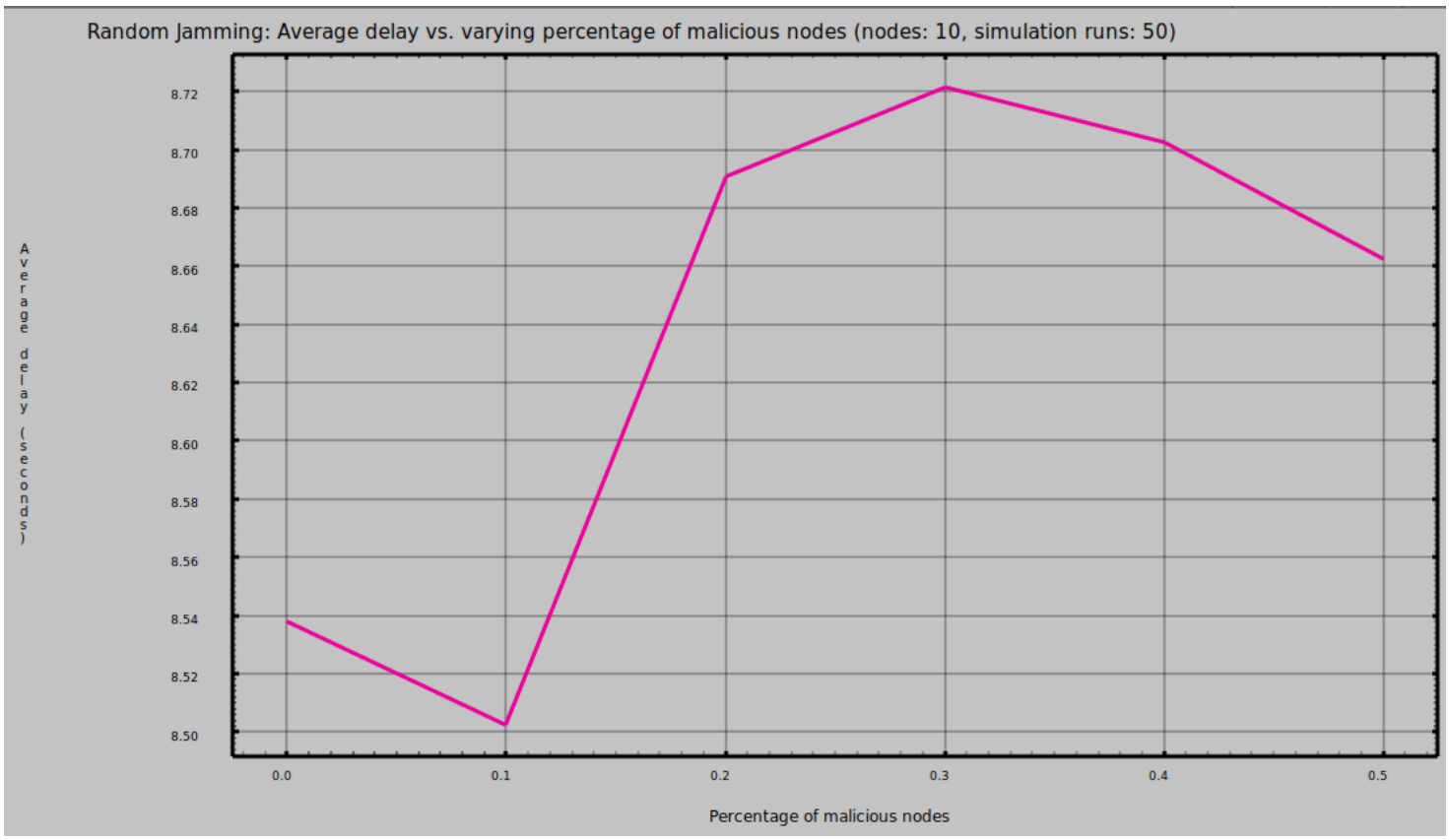


Periodic Jamming: Average delay vs. varying percentage of malicious nodes (nodes: 10, simulation runs: 50)

Random Jamming:



Random Jamming: Average throughput vs. varying percentage of malicious nodes (nodes: 10, simulation runs: 50)



Random Jamming: Average delay vs. varying percentage of malicious nodes (nodes: 10, simulation runs: 50)

As we can see from the above graphs, the throughput decreases and the delay increases as the percentage of malicious jammers nodes increases.

From this, we can conclude that the presence of jammer nodes cause a performace hit on the network.

Note: All these files can be accessed online from https://github.com/PramodJose/ns2-malicious-behaviour