COL334/672: Assignment 3 – option B

Attached is a trace of Bluetooth encounters seen by devices which were running an application that periodically probed for the presence of other Bluetooth devices. The file **proximity-edges-timestamps.csv** contains 3 columns:

- Timestamp when device-id-1 saw device-id-2
- Device-id-1
- Device-id-2

1700+ Bluetooth devices were sighted over a cumulative 250,000+ sightings during a four to five day conference setting.

You can envision this as a time-evolving graph where edges form spontaneously between pairs of nodes when they come into contact, during which instance they could exchange data. The devices may then move and pair up with other nodes with whom they could exchange data. We want to simulate different routing algorithms over this network to enable nodes to broadcast information to all other nodes, and to send information to some particular nodes.

Assume each sighting to be sufficiently long to be able to send a chunk of data.

To experiment with different routing algorithms, you will need to design a discrete event driven simulator as follows:

- Maintain an array of all devices, with each element containing a data structure for the state of that device at the current time.
    - For example, you may need to maintain state for whether or not the device has received the chunk, or you may even need to maintain state of a list of all the neighbours to which the device forwarded the chunk or received the chunk from
- Start reading the trace file line by line
- On each sighting, take suitable action on the pair of devices to update their respective states.
    - For example, if device-id-1 has the chunk and according to the routing algorithm being used it decides to forward the chunk to device-id-2 then you should update the state of device-id-2 of having received the chunk
    - Similarly, if each device maintains a list of its neighbours with whom it has exchanged the chunk, then you should update this list upon each sighting when the chunk was exchanged
- On each sighting, also update global variables like the number of devices which have received the chunk, and use this to test for simulation termination conditions

Now do the following experiments:

1. Start with building a simple broadcast algorithm that runs on each device. The algorithm takes as a parameter K the number of times a device can forward a chunk, and each device forwards the chunk to the first K neighbours it sights. It does not forward the chunk to devices which have already received the chunk in the past from that device or

from some other devices – such a protocol can be built easily where upon contact, the pair of devices first query each other if they already have the chunk, and if not only then the device with the chunk sends it to the device which has not received the chunk so far.

a. Starting with device 26, find the time by when the chunk reaches 90% of the devices for K=1, 2, 3, 4, 5… If for small K the chunk does not reach 90% of the devices then mention the percentage of devices that it does reach by the end of the sightings trace. Build a graph with K on the x-axis and time-taken-for-broadcast on the y-axis. Is there a value of K beyond which there is no more noticeable improvement in the time taken for broadcast? Why would this be the case?
b. How many copies of the chunk are made in each run of the broadcast?
c. Now choose 100 devices randomly and run 100 experiments, one for each device as the source of the broadcast. Build a graph with K on the x-axis and the mean time (calculated across the experiments) on the y-axis with standard deviation error bars around each mean. This is a more statistically robust way to ensure that the trends noticed in part (a) are persistent.

2. Let us make the algorithm more interesting. As part of a pre-classification exercise, run through the entire data and find the degree of the devices – the degree is the count of unique neighbours the device sees in the trace. Plot the degree distribution as a CCDF on a log-log scale. Divide the devices into three classes – top S% of devices in terms of degree as super-nodes which are the most mobile and come in contact with most devices, bottom L% of devices which are the least mobile and will require special treatment to be reached, and the remaining devices which are the middle class ordinary nodes. Now, upon a sighting, instead of sending the chunk to the first K neighbours a device sees, a device sends the chunks to super-nodes it encounters with a different probability than to the ordinary nodes, and always with probability 1 sends the chunk to the least mobile devices as a special treatment.
    a. Start with S = 0.5% and L = 70%, and vary the transmission probability to super-nodes from being equal to the transmission probability to ordinary nodes (50%, 50%), to extreme variations like being almost 100 times the transmission probability to ordinary nodes (99%, 1%), or almost 1/100 times the transmission probability to ordinary nodes (1%, 99%)[1]. As before, start with node 26 and plot the time taken for broadcast against the probability of transmission to super-nodes. Also plot the average number of transmissions made by the super-nodes, by the ordinary nodes, and by the least connected nodes. Report your observations in terms of which combinations are the best that the broadcast also completes quickly as well as not many of the super-nodes or ordinary nodes are

[1] A combination of (x%, y%) means that upon encountering a super-node the device will with x% probability send a chunk to the node. In the simulator implementation, you can generate a random number between 0-1 and if it is greater than x% then decide to transmit else do nothing. Similarly, upon encountering an ordinary node the device will with y% probability send a chunk to the node. Note that we have simply restricted the sample space for (x%, y%) by having y=1-x in these proposed experiments, but you should feel free to experiment in other sample spaces too if you want.

stressed out too much by having to make many copies. Do you see a tradeoff between these two goals? Why does the tradeoff matter?

b. Repeat the experiment with 100 randomly chosen origin nodes, and plot the mean and standard deviation across the experimental runs to determine whether the same trends persist statistically.
c. Now vary S to 1%, 5%, 10%, etc and see how the graphs and ideal configurations change. Explain your observations.
d. Similarly, vary L to see how the graphs change, and explain your observations.

3. Now let us use a different strategy. The file **modularity-class.csv** contains a mapping of the devices to about 10 different community classes. These classes were obtained using the Gephi graph analysis software, to detect tightly knit communities which have a lot more interconnections between their nodes than with nodes in other communities. The decision of whether to transmit or not transmit a chunk can be dependent on if the recipient device is in the same community class or not, to balance between dedicating effort to spreading the chunk within a community Vs pushing it outside a community. As before, define a probability X% of allowing transmission outside a community class and a probability Y% = 1 - X% of allowing transmission within a community class. Experiment with different combinations of (X%, Y%) to find the ideal combinations which complete the broadcast quickly as well as do not stress out some nodes with having them make very many copies. Contrast this with the other methods we have tried earlier.

What to submit:

i. /src containing a script (in python or perl or any other language you are comfortable with) for part 1, and a README file, that takes the number of times a chunk can be forwarded (K) as input and the source node, and produces as output the various information asked in part 1 in neat CSV files. These CSVs will be used to plot the graphs asked. You should also submit scripts you may write to run the simulation for different values of K and the source nodes, and save the data in output files that can be plotted using software like gnuplot.
ii. /src containing a script (in python or perl or any other language you are comfortable with) for part 2, and a README file, that takes inputs in the order S% (percentage of super nodes), L% (percentage of least mobile nodes), transmission probability to super nodes, transmission probability to least mobile nodes (note that you can also experiment with varying sample spaces for (x%, y%) as mentioned), and the source node. You should also submit scripts for running the simulator for different parameter values. It should produce as output the various information asked in part 2 in neat CSV files. These CSVs will be used to plot the graphs asked.
iii. /src containing a script (in python or perl or any other language you are comfortable with) for part 3, and a README file, that takes as arguments probability of transmission outside community (X%) and probability of transmission within the community (Y%), and produces as output the various information asked in part 3 in neat CSV files. Also submit scripts to run the simulator for different values of the parameters. These CSVs will be used to plot the graphs asked.
iv. /doc containing a pdf report on the data, analysis, and insights asked in parts 1-3. The report should contain all of the plots asked in the questions, with clear

explanation of the insights – presentation has a high premium to clearly convey your insights, as usual. Please make sure that the plots are clear and are self-explanatory. Corresponding to each question's answer, clearly explain your observations as asked in the reports.

Please use the files **proximity-edges-timestamps.csv and **modularity-class.csv** for all of your analyses. Your programs should read these files only to produce the required results, and not any file of your choice.