Name: Rajib Dey

PID: 4166566

CAP 6135, Spring 2018

Programming Project 3: Internet worm propagation simulation

1. Simulate a random-scanning worm propagation:

Designing the simulation:

a) I designed the java program to randomly check any IP address between 1 and 100,000. This is done by generating a random number. If the generated random number falls in between the given vulnerable IP list, then it is tagged as infected otherwise it is tagged as safe. A few essential coding approaches is described below:

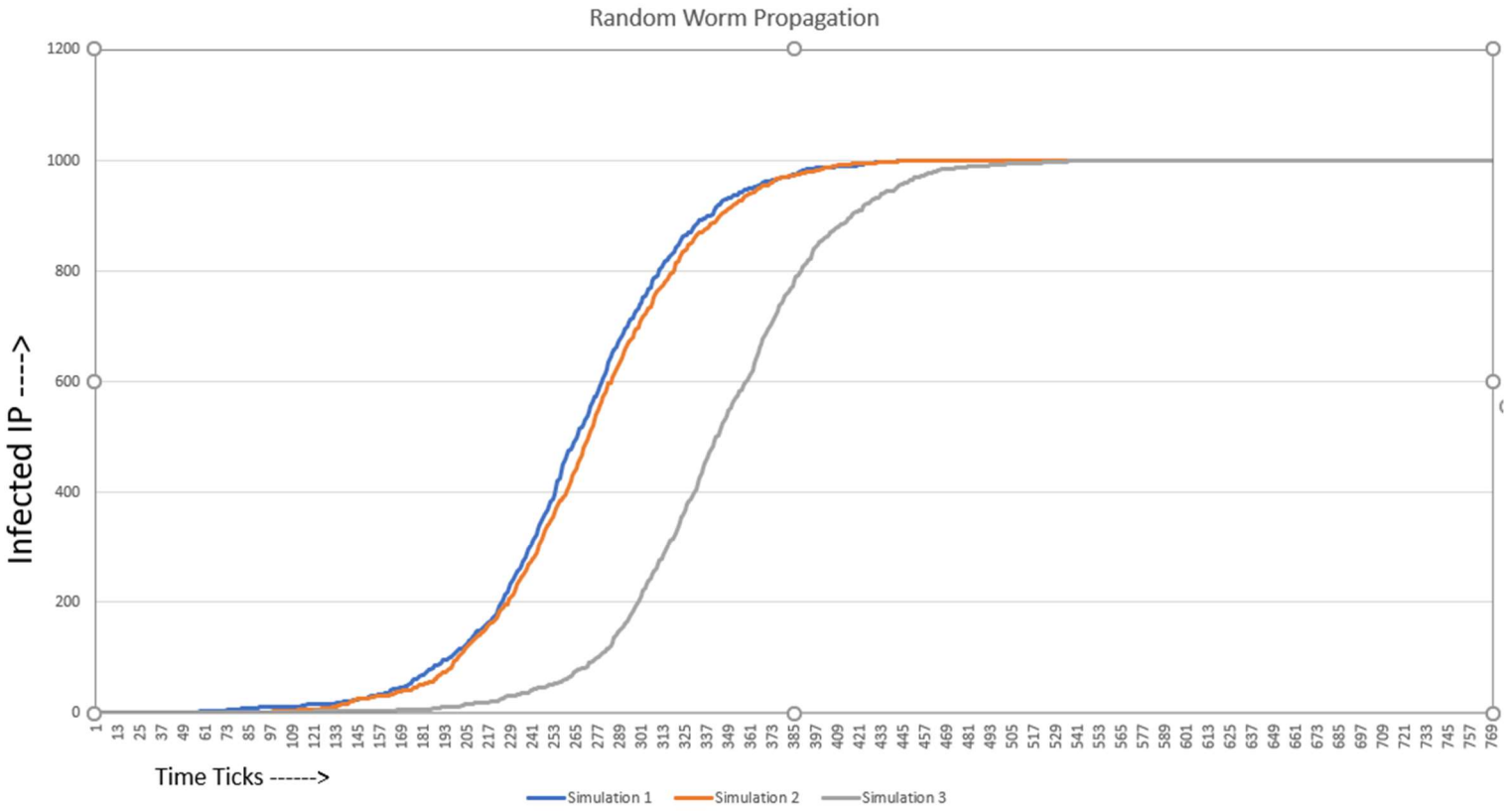
- I hardcoded a high value for the maximum time tick as 2500
- To store the number of all the infected IP I used "results[maxTime+1][4]". This array index is maintained starting from 1.
- To store the status of the infected IP, 100,000 objects named nodeStatus[] has been created of enum Node type.
- Vulnerable IPs are identified according to the given instructions.
- To count the number of infected IPs in priorNodeStatus, count is initialized
- The *while* loop will loop for maxTime time ticks (maxTime in this case is 2000).
- Inside *while* loop there are two *for* loops. The first for loop is used to count the number of IPs in the previous time tick. In the second for loop every infected IPs will search for three other IPs. If the searched IP falls inside the vulnerable list then it is tagged as infected and increase the infected count for that time tick. It will store the index value of the infected IP in the priorNodeStatus so that it can join the infected IP list in the next time tick.
- Used a result matrix to store the infected count.
- This simulation was done three times and the results were stored in a text file.

b) According to the program I wrote:

Total Time Tick for simulation #1 to infect all vulnerable IP is = 482 tick
Total Time Tick for simulation #2 to infect all vulnerable IP is = 445 tick
Total Time Tick for simulation #3 to infect all vulnerable IP is = 569 tick

Because of the unpredictable nature of randomness, each time you run the program it will give you a different result.

Plot:



The data from the text file is imported in an excel file (submitted along with this report) and used to plot the above graph.

At around time tick 200 (about 20% infections for all the simulations) we can notice an exponential increase for all 3 simulations. At around 300 time ticks the infection is almost about 80% then the growth slows down. Which we can see in the form that at 350 time ticks it reaches only about 90%. At 569-time tick, they all achieve 100% infection at least in this case.

2). Simulate a local-preference scanning worm propagation:

Designing the simulation:

a. I designed the java program to randomly check any IP address between 1 and 100,000. This is done by generating a random number. If the generated random number falls in between the given vulnerable IP list, then it is tagged as infected otherwise it is tagged as safe. A few essential coding approaches is described below:

- I hardcoded a high value for the maximum time tick as 2500
- To store the number of all the infected IP I used "results[maxTime+1][4]". This array index is maintained starting from 1.
- To store the status of the infected IP, 100,000 objects named nodeStatus[] has been created of enum Node type.
- Vulnerable IPs are identified according to the given instructions.
- To count the number of infected IPs in priorNodeStatus, count is initialized.
- The *while* loop will loop for maxTime time ticks (maxTime in this case is 2000).
- Inside *while* loop there is a *for* loop. At every time tick, it runs for omega number of times. A random variable named probability is created for every infected IP, which can have a float value between 0 to 1. If the value of the variable is between 0 and 0.8, the infected computer will search for 3 random values (Eta=3 is given) in the range of x-10 and x + 10. Here x is denoted as the infected computer. As per local preference principle, 3 vulnerable IPs will be assigned as infected. Otherwise, it will search for IPs in the same way as it was doing in random allocation. If the searched IP is vulnerable it will infect it and increase the infected count for that time tick. It will store the index value of the infected IP in the priorNodeStatus so that it can join the list of infected IPs in the next time tick.

- Used a result matrix to store the infected count.
- This simulation was done three times and the results were stored in a text file.

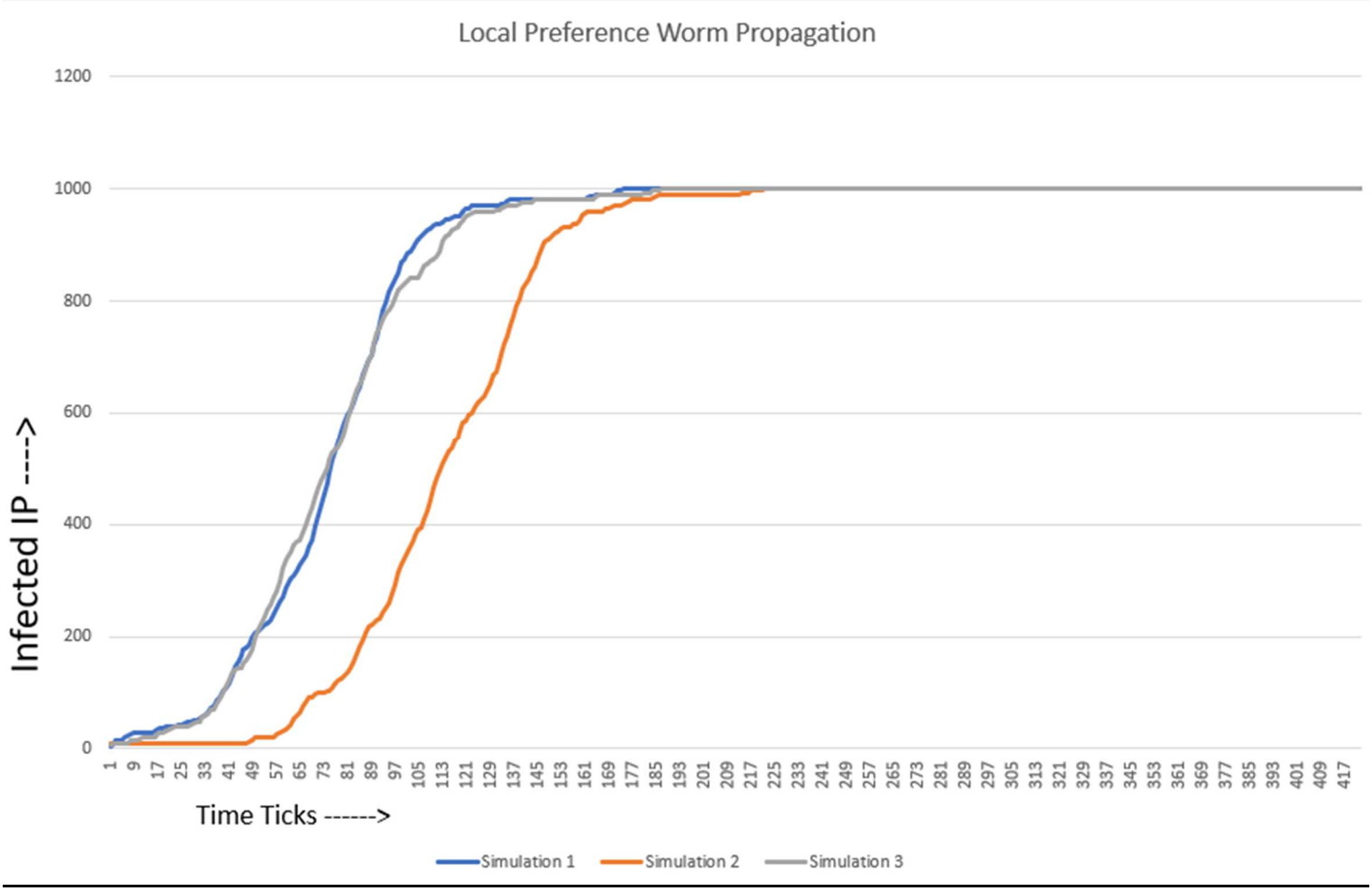
b.

According to the program I wrote:

Total Time Tick for simulation #1 to infect all vulnerable IP is = 174 tick
Total Time Tick for simulation #2 to infect all vulnerable IP is = 222 tick
Total Time Tick for simulation #3 to infect all vulnerable IP is = 187 tick

Because of the unpredictable nature of randomness, each time you run the program it will give you a different result.

Plot:



The data from the text file is imported in an excel file (submitted along with this report) and used to plot the above graph.

We can notice an exponential increase for all 3 simulations. The 3rd simulation has lesser growth than the other two. For the first 2 simulations, at around 50 time tick they achieve 20% infection. Then at around 90 time tick they achieve 80% infection. They further achieve almost 90% infection within 100-time tick, which is an exponential growth. So ultimately, a typical S curve is being generated exhibiting the exponential growth rate.

If we compare between these two worm propagation method, we can clearly state that the local preference scanning is much more efficient that the random one. For random scanning, the infection rate reaches 90%-100% in around 445 time ticks. While Local Preference scanning achieves the same in around 174 Time ticks.

Compile and run instructions:

To run the programs that are implemented in Java (random.java and localpref.java) in the Eustis machine, please follow the following instructions:

1. Open a terminal and *cd* to the directory where all the files are located, eg: cd path/to/file/code/
2. To compile use this command *javac random.java*
3. Then enter the following command to run the program: *java random*
4. Two text files (random.txt and localpref.txt) will be generated. These two files are used to generate the graphs using Microsoft Excel.

Screenshot

```
ra295724@net1547:~/CAP6135/Assign3$ ls
localpref.java  random.java
ra295724@net1547:~/CAP6135/Assign3$ sudo apt-get install
Those are probably all already installed.
ra295724@net1547:~/CAP6135/Assign3$ javac random.java
ra295724@net1547:~/CAP6135/Assign3$ ls
localpref.java  Node.class  random.class  random.java
ra295724@net1547:~/CAP6135/Assign3$ java random
Total Time Tick for simulation #1 to infect all vulnerable IP is = 482 tick
Total Time Tick for simulation #2 to infect all vulnerable IP is = 445 tick
Total Time Tick for simulation #3 to infect all vulnerable IP is = 569 tick
ra295724@net1547:~/CAP6135/Assign3$ javac localpref.java
ra295724@net1547:~/CAP6135/Assign3$ ls
localpref.class  localpref.java  Node.class  random.class  random.java  random.txt
ra295724@net1547:~/CAP6135/Assign3$ java localpref
Total Time Tick for simulation #1 to infect all vulnarable IP is = 174 tick
Total Time Tick for simulation #2 to infect all vulnarable IP is = 222 tick
Total Time Tick for simulation #3 to infect all vulnarable IP is = 187 tick
ra295724@net1547:~/CAP6135/Assign3$ ls
localpref.class  localpref.java  localref.txt  Node.class  random.class  random.java  random.txt
ra295724@net1547:~/CAP6135/Assign3$
```

File-List and Description:

The submitted folder contains

1. This report in pdf format
2. *Code* folder contains: Java files *localpref.java* and *random.java* . These two are the main program that I implemented.
3. *Data and Graph* folder contains:
    a. Two txt files that was generated after running the programs. *Localpref.txt* and *random.txt.* These two contains data about how many IPs were infected in which time tick and in which time simulations.
    b. Two excel files. Namely *localpref.xlsx* and *random.xlsx* . These two were generated using the abovementioned txt files to generate the graphs for both worm propagation systems.
4. *Screenshots* folder contains three image files
    a. *Local Preference Worm Propagation graph.PNG* : This is the graph for the Local preference worm propagation.
    b. *Random Worm Propagation graph.PNG*: This is the graph for the Random worm propagation.
    c. *Screenshot.PNG*: This is the terminal screenshot showing the code compiling and running in the *Eustis* machine. The *ls* command shows all the files that were generated after the compilation and after the programs were done executing.