Discrete-Time Simulation of Worm Propagation

Bhoomi Patel
CAP 6135
3/31/2019

1. Introduction

The goal of this assignment is to simulate simple worm propagation in a medium-scale network by using discrete-time simulation. Two types of worm propagation methods are simulated in this assignment:

1. Random-scanning worm propagation

2. Local-preference scanning worm propagation

Both processes consist of an infected computer scanning different IP addresses to find vulnerable targets, and then infect those targets with a worm. These propagation methods are similar to that of the Code Red worm, which is a non-topological malware. It means that any infected node can reach any other node in a single hop in a network.

2. Simulator Design and Code

The solution to this assignment is written in Python for two reasons. First, Python's `random` and `numpy` modules allow making a selection from a given list based on the selection probabilities assigned to each item in a list. This is helpful for the simulation of a local-preference scanning propagation method, in which the IP addressed to be scanned is selected based on an 80-20 probability split. Second, Python also provides modules to plot data on graphs, which help with visualizing the rate of infection.

The following parameters are provided for this assignment:

$\Omega = 100000$; address space or number of IP addresses in the network

$N = 1000$; total vulnerable computers

$\eta = 3$; scan rate or scans per infected host

The assignment also states that three simulations of each type of worm propagation method should be performed. The runtime, or total number of ticks to be simulated in each simulation run, is to be specified by the program author depending on how long each simulation typically takes to finish infecting all 1000 computers. The runtimes selected for the two propagation methods are:

- 700 ticks for random-scanning technique
- 300 ticks for local-preference technique

The vulnerable addresses are the first ten IP addresses of every 1000*n address block, where n = 0, 1, 2, 3…99. Few examples of the vulnerable addresses would be:

[1, 2, 3… 10]

[1001, 1002… 1010]

[2001, 2002… 2010]

[99001, 99002 … 99010]

For each of the three simulation runs, the first step of the program is to initialize the IP address space, and assign the status of “infected”, “immune”, and “susceptible” to the appropriate IP addresses. At time tick 0, only address 1001 is “infected”.

Each infected computer performs $\eta = 3$ scans every time tick in an effort to locate susceptible computers to infect. The scanning process consists of generating random IP addresses and checking their node status, i.e. whether they are infected, immune, or susceptible.

The process of IP address generation for the random-scanning technique is very straightforward. The following command is used to generate a random integer between 1 and 100000 which then becomes the IP address that will be scanned.

```
ip = random.randint(1, 100000)              # Select random IP address: 1 <= ip <= 100000
```

For the local-preference scanning technique, the process has a few extra steps. First, one random IP is generated using the `random.randint()` function. Next, a second, local, IP address is selected from range of addresses in the neighborhood of the infected computer performing the scans, i.e. 10 addresses to the left and 10 to the right of the computer performing the scans.

It is between these two IP addresses that the program, or the “infected computer”, must choose to scan. The probability of the local IP address being selected is 0.8 and the probability of the random IP address being selected is 0.2. The function `np.random.choice()` takes these options and the probability distribution and returns a list containing one element, which is the IP address to be scanned. That list is stored in `ip`, and `ip[0]` gives the IP address value. The steps discussed above are coded in Python as shown below:

```
# Select random IP address: 1 <= ip <= 100000
random_ip = random.randint(1, 100000)
# Select IP local to infected computer with IP x
local_ip = random.randint(infected_ip - 10, infected_ip + 10)
#####################################################################
# We want the scan to select the generated local_address with a probability of 0.8
# and the generated random address with a probability of 0.2
# The command below will do so (select IP to scan):
ip = np.random.choice([local_ip, random_ip], size=1, p=[0.8, 0.2])
```

A crucial aspect of the design process is to ensure that a newly-infected computer does not itself start infecting other susceptible computers until the time tick *after* the one it is infected in. To do this, the program makes a note of all the computers infected at the start of a new time tick. Only the computers that are infected going into the time tick are allowed to scan random IPs to search for susceptible computers.

```
# Find out which computers were infected coming into this time tick:
currently_infected = [1001]
for i in range(0, 100):
    for j in range(1, 11):
        if NodeStatus[j+i*1000] == "infectious":
            currently_infected = currently_infected + [j+i*1000]
```

This step ensures that the newly affected computers will not be performing any scans of their own until *after* the current time tick has ended and until a new `currently_infected` list is generated in which they are included.

3. Results

Few things to keep in mind as the results are discussed. A single simulation run ends when all 1000 vulnerable or "susceptible" computers have been infected by the worm. Because of the amount of randomness involved in the program, these results are subject to change in each execution, however they will not vary greatly.

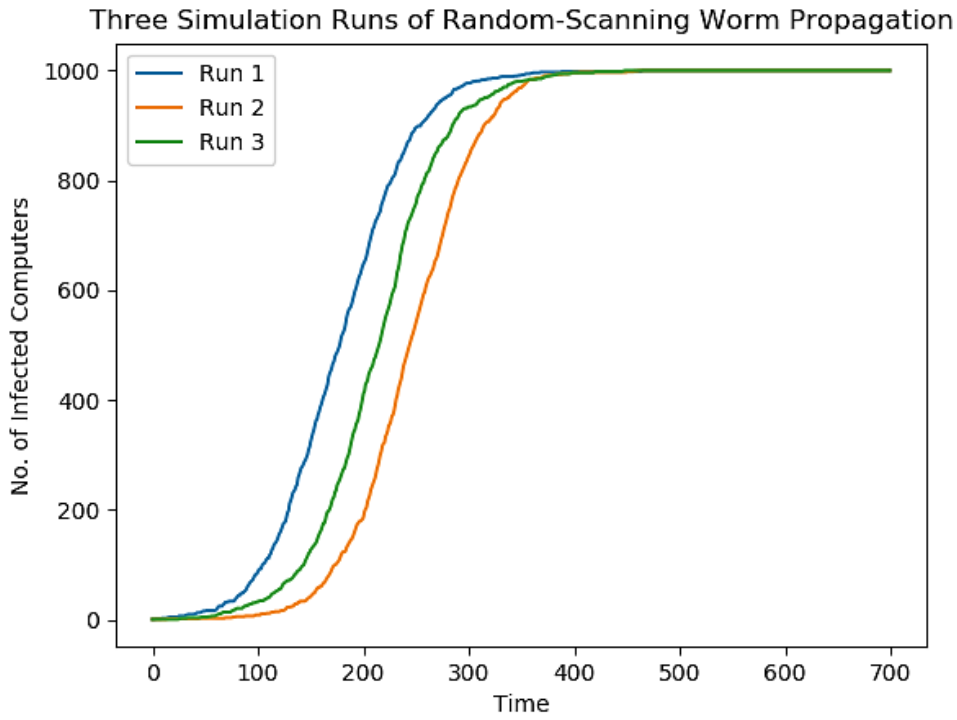
3.1 Random-Scanning Worm Propagation

The figure below shows after how many simulated ticks each simulation of the random-scanning worm propagation ends.

```
bh461410@net1547:~/Project3$ python3 Simulator1.py
Simulation 1 ended in  465 ticks
Computers infected:  1000
Simulation 2 ended in  461 ticks
Computers infected:  1000
Simulation 3 ended in  448 ticks
Computers infected:  1000
All simulations completed.
```

Out of a runtime of 700 ticks, simulation 1 ended after 465 ticks, simulation 2 ended after 461 ticks, and simulation 3 ended after 448 ticks. All simulations generally ended in under 500 ticks.

The graph below shows the rate at which each susceptible computer is infected with respect to time ticks when using the random-scanning worm propagation technique.

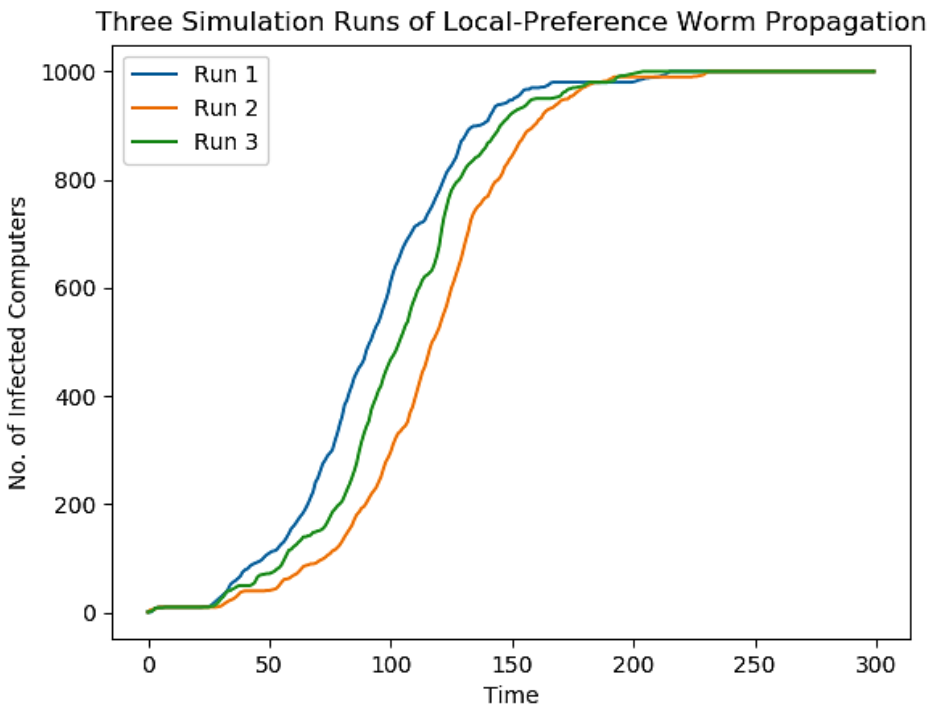


3.2 Three Simulation Runs of Local-Preference Worm Propagation

Out of the runtime of 300 ticks, simulation 1 ended after 215 ticks, simulation 2 ended after 231 ticks, and simulation 3 ended after 204 ticks. All simulations generally ended in under 250 ticks.

```
bh461410@net1547:~/Project3$ python3 Simulator2.py
Simulation 1 ended in  215 ticks
Computers infected:  1000
Simulation 2 ended in  231 ticks
Computers infected:  1000
Simulation 3 ended in  204 ticks
Computers infected:  1000
All simulations completed.
```

The graph below shows the rate at which each susceptible computer is infected with respect to time ticks when using the local-preference worm propagation technique.



3.3 Observations

It can be observed from the two graphs in the previous sub-sections, that the local-preference method of worm propagation is faster than the random-scanning method. The explanation is as follows:

All the vulnerable computers in the created network are assigned consecutive IP addresses in blocks of 10. This means that at any given time tick, an infected computer is likely to be "close" to a susceptible computer. In addition, only vulnerable computers can be infected by a worm. Therefore, during scanning, when a probability of 80% is assigned to the selection of an IP address local to an infected computer, it is only logical that the vulnerable IP addresses are more likely to be selected, and thus infected at a much faster rate.