CSC 790 A: Project 2
Landstreicher hat Würmer gern
Fall 2014

Due
2:00pm Thursday, 3/6

Project Description

Internet worms are a form of malware that spreads via the network with little human assistance. The success of an Internet worm is largely due to the ability to quickly identify vulnerable hosts. The paper by Staniford et al. (reference at the course web site) describes different methods to find vulnerable hosts, this project will investigate the performance of these propagation methods using simulation.

Worm Simulator

The tar file `wormSim-2014.tar` located at the course web site contains a simple discrete event simulator for a network of stations (hosts). The archive consists of the five files: `driver.cpp` contains the `main` function of the simulator, `station.h` defines a `Station` object that models a host, `defs.h` defines simple constants, `event.h` defines an event in the simulator, and `eventq.h` defines the *calendar* of events.

A discrete event simulator operates by removing events from a calendar then performing the associated task(s) for that event. This is typically done in the `main()` function using a `while` loop that removes the next event (which is the current event) from the calendar then calls the correct functions for this current event (see the `driver.cpp` file). Therefore the calendar is modeled as an ordered (according to time) queue of events which is defined in the `eventq.h` file.

There are two basic events for this simulation, `Propagate` and `Infect`. The `Propagate` event occurs when an infected station attempts to find another vulnerable station. For this event the `main()` will have the appropriate `Station` object call the `propagate()` function. This function will select another station (**you will add the majority of your code in this function**) and insert an `Infect` event in the calendar. The `Infect` event causes the `infect()` function to be called. If the `Station` associated with this call is vulnerable, then it will insert a `Propagate` event in the calendar, which will cause this `Station` begin infecting other stations... *and the evil continues.*

The simulator is designed to stop after all the vulnerable stations have been infected. The simulator will continue to run however, until all the `Events` already inserted in the calendar have been processed. Do not change this behavior, since the number of infection attempts will be used to compare station selection methods.

The simulator will produce 2 files per execution, `data.dat` and `infect.dat`. The file `data.dat` contains two columns of data, the first column is the time (in seconds) while the second column is the number of infected stations. The file `infect.dat` has three column: the first column is the station index, the second column is the time when initially infect, and the third column is the number of infection attempts by other stations (how many times did other stations attempt to infect this station).

Modeling the Network

This project will simulate an IP class B network, which is the same type of address space given to Wake Forest University. Class B addresses, for example `152.17.0.0/16`, allow the last two bytes in the IP address to be locally defined. Let's assume this address space is subnetted where the third byte is used to define a subnet and the fourth byte is used to define a host. These bytes can be any value between 1 and 254. (*yes there are* 2^8 *possible values per byte, I think 0 and 255 should be reserved, just humor me*).

This address space can be viewed as a 254×254 matrix, for example using the address `152.17.`$x.y$, the x is the row and y is the column. We will assume every legal address in the `152.17.0.0./16` has a station running, therefore there are $254 \times 254 = 64,516$ stations in the simulator. Although the network address space can be viewed as a matrix, it is more easily managed in the simulator as a list (as seen in the `driver.cpp` file). As a result we need to translate a list index into the proper row and column index, this simple translation in provided by the `Station` object.

Assumptions

There are some important assumptions for this simulator. As already described, it is assumed every legal address in the class B network has a station. This results in $254 \times 254 = 64,516$ stations, but you may wish to use a smaller number for debugging your simulator. As defined in the `main()` function, initially only 25% are vulnerable (you will change this value depending on the project question).

Note, the **time in the simulator is recorded in seconds**. There are two important random variables for the simulation, the time to attempt an infection and the time between propagation attempts. The time between infection is the amount of time required for an infection attempt (for example network and computer processing delay). For this simulation this time has a uniform distribution between 1 and 200 milliseconds. The time between propagation represents the time between station selection (for example worm processing delay). For this simulation, this time has a uniform distribution between 10 and 38 milliseconds.

---

1 Random Station Selection

For this part of the project the worm will randomly select a station (excluding itself) within the class B address range. Assume 25% of the stations are vulnerable.

Questions (*10 points*)

You must also correctly answer the following questions to receive full credit for this part of the project.

1. Describe how the worm randomly selects a station (show your and explain your code).
2. Plot the number of stations infected as time increases (use hours for time). Describe the performance of the station selection method.
3. Plot the time until initial infection per vulnerable station. This must be a 3 dimensional plot, where the x and y axes are the address bytes, and the z axis is the infection time. Only vulnerable stations should be plotted. Describe the time required per station. Is the time the same for stations? What is the average and standard deviation infection time? Comment on your results.
4. Plot the number of infection attempts per vulnerable station. This is the number of times other stations have attempted to infect a vulnerable station. This must be a 3 dimensional plot, where the x and y axes are the address bytes, and the z axis is the number of attempts. Only vulnerable stations should be plotted. Describe the time required per station. Is the number of infection attempts the same for stations? What is the average and standard deviation number of infection attempts? Comment on your results.
5. Change the vulnerable percentage to 50% and repeat the second question. Is there a difference between the percentages? Why or why not?

As an example, consider a system where the maximum value of the last two address bytes is 20. The graphs required for the first four questions are given in figure **??**.

---

2 Fletcher Worm, Limited Effort Station Selection

For this part of the project, worm will randomly select a station (excluding itself) within the class B address range; however, after attempting x infections (does not matter if the attempt was successful) the station will stop propagating. Assume 25% of the stations are vulnerable.

Questions (*15 points*)

You must also correctly answer the following questions to receive full credit for this part of the project.

1. Describe how the worm randomly selects a station. What is the minimum value of x that will infect all vulnerable machines?
2. Using the minimum x value, plot the number of stations infected as time increases (use hours for time). Describe the performance of the station selection method.

3. Using the minimum x value, plot the time until initial infection per vulnerable station (3 dimensional plot). Is the time the same for stations? What is the average and standard deviation infection time? Comment on your results.

4. Using the minimum x value, plot the number of infection attempts per vulnerable station (3 dimensional plot). Describe the time required per station. Is the number of infection attempts the same for stations? What is the average and standard deviation number of infection attempts? Comment on your results.

---

3 Topological Station Selection

For this part of the project the worm will perform a topological selection (excluding itself) within the class B address range. The worm should randomly select a station within its subnet 75% of the time (excluding itself) and randomly select a station outside of the local subnet 25% of the time. Assume 25% of the stations are vulnerable.

Questions (*15 points*)

You must also correctly answer the following questions to receive full credit for this part of the project.

1. Describe how the worm randomly selects a station.

2. Plot the number of stations infected as time increases (use hours for time). Describe the performance of the station selection method.

3. Plot the time until initial infection per vulnerable station (3 dimensional plot). Describe the time required per station. Is the time the same for stations? What is the average and standard deviation infection time? Comment on your results.

4. Plot the number of infection attempts per vulnerable station (3 dimensional plot). Describe the time required per station. Is the number of infection attempts the same for stations? What is the average and standard deviation number of infection attempts? Comment on your results.

---

4 Hit List

For this part of the project the worm will use a *hit list* that defines all hosts within the class B address range. Starting with the first station on the list (excluding itself) the worm should attempt to infect a station. If the infection is successful the worm should divide its list in half, and use the first half and instruct the newly infected station to use the second half. Using this approach, **every station should have only one infection attempt**. Assume 25% of the stations are vulnerable.

Questions (*25 points*)

You must also correctly answer the following questions to receive full credit for this part of the project.

1. Describe how the worm randomly selects a station.

2. Plot the number of stations infected as time increases (use hours for time). Describe the performance of the station selection method.

3. Plot the time until initial infection per vulnerable station (3 dimensional plot). Describe the time required per station. Is the time the same for stations? What is the average and standard deviation infection time? Comment on your results.

4. Plot the number of infection attempts per vulnerable station (3 dimensional plot). Describe the time required per station. Is the number of infection attempts the same for stations? What is the average and standard deviation number of infection attempts? Comment on your results.

---

5 Worm Comparison

For this part of the project, you will compare the performance of the previous selection methods.

Questions (*15 points*)

You must also correctly answer the following questions to receive full credit for this part of the project.

1. Graph the number of stations infected as time increases (use hours for time) for all the selection methods. Assume 25% of the stations are vulnerable. Describe the performance of the station selection methods. Which method is best, which is worst, and explain why?

---

6 Worm Fight Club

Recall the cheese worm was an attempt to make a friendly worm that would patch vulnerable systems and possibly smoke your stash (*not so friendly*). This part of the project will determine if releasing a good worm that patches systems is a reasonable defense. You can assume both worms (good and bad) use identical selection methods and are released at the same time. In addition, for each station in the network identify which worm contacted it first (good or bad).

Questions (*20 points*)

Although we all know the first rule in worm fight club, you must also correctly answer the following questions to receive full credit for this part of the project.

1. Graph the number of stations infected as time increases (use hours for time) for all the selection methods. On the same figure also graph the number of patched systems as time increases. Assume 25% of the stations are initially vulnerable.

2. Run this experiment several times and record the number of infected of stations. Then create a bar chart that shows the percentage of infected hosts (x-axis) as compared to the percentage of experiments that ended with that percentage of infected hosts (y-axis). For example you may observe that 18% of the experiments ended with 10% of the stations infected. What is the average number of infected machines? Is this what you expected? Why or why not?

Solutions

1.1

In the `station.h` file, we changed the codes as follows:

```
void propagate(double time, EventQueue& eQueue)
        {
                int toID;               // find potential victim
                int x;
                int y;
                do{
                        x = rand()%MAX_STATIONS;        // address is 152.17.x.y
                        y = rand()%MAX_STATIONS;        // address is 152.17.x.y
                        toID = x*MAX_STATIONS + y;
                        }while( toID == id_ );  // selection excluding itself

    ...
    }
```

As you can see, x and y will be randomly assigned a value between 0 and 253, this is a 1-to-1 mapping to 1 254, besides the station vector object s is also defined from 10 to $254 \times 254 - 1$, so this doesn't affect the simulation. Also, the *do...while* loop will ensure the worm will select a station excluding itself.

1.2

From `data.dat` we have

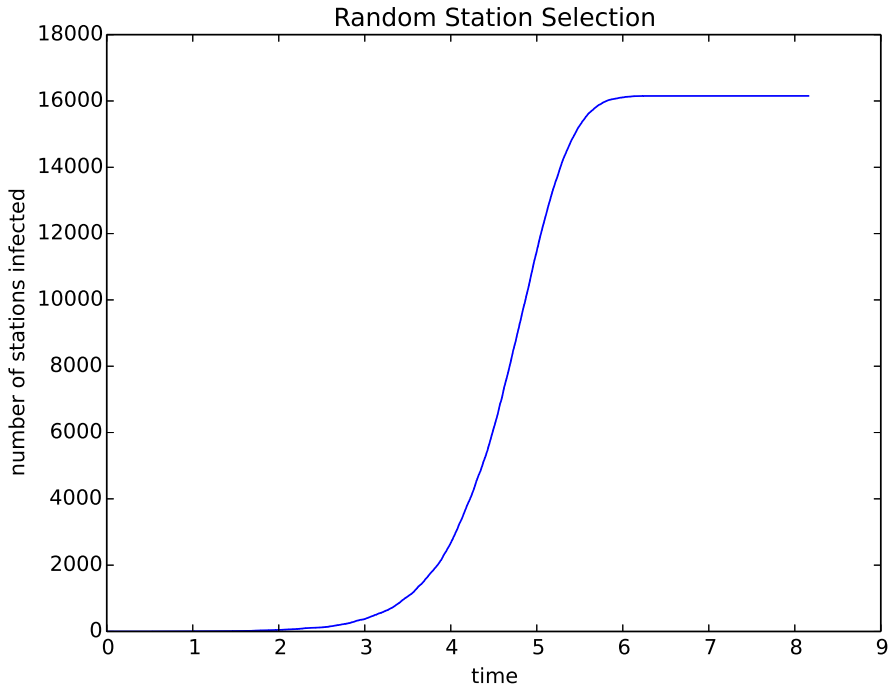

Figure 1: Number of Infected Hosts using Random Station Selection

By the random selection method, at the beginning of propagation step, the number of infected hosts grows slower, when there are sufficient infected hosts, there is a small burst time that the number of infected hosts jumped very high, as you can that the curve is very steep in the graph.

1.3

For `infect.dat` we have

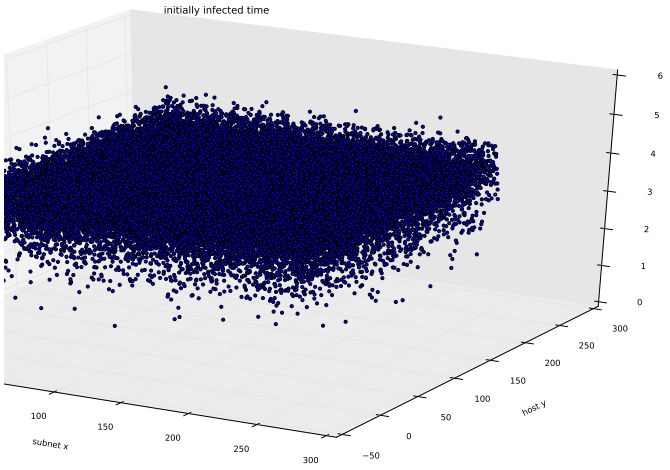

Figure 2: Initially Infected Time using Random Station Selection

No, the time is not quite the same for stations, the average is 3.6517 and the standard deviation is 0.4951 for the infect time. Since the standard deviation is relatively small, which means most of the station get infected around almost the same time.

1.4

For `infect.dat` we have

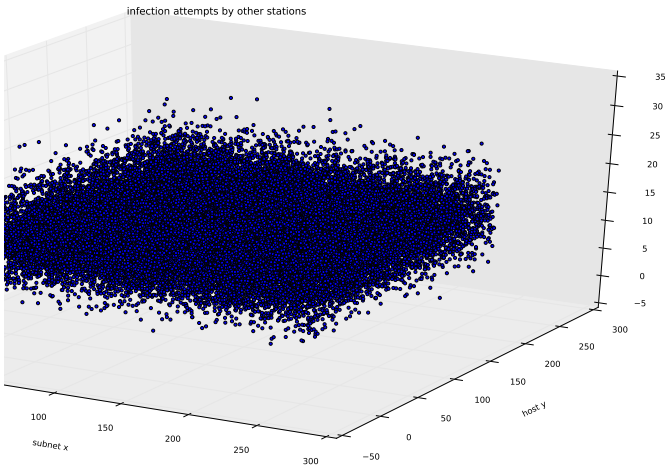

Figure 3: Attempts by other Stations using Random Station Selection

No, the attempts is not quite the same for stations, the average is 13.23 and the standard deviation is 3.6371 for the number of the infection attempts. Here we can see that almost all vulnerable hosts got multiple infection attempts by other infected hosts.

1.5

For `data.dat` we have

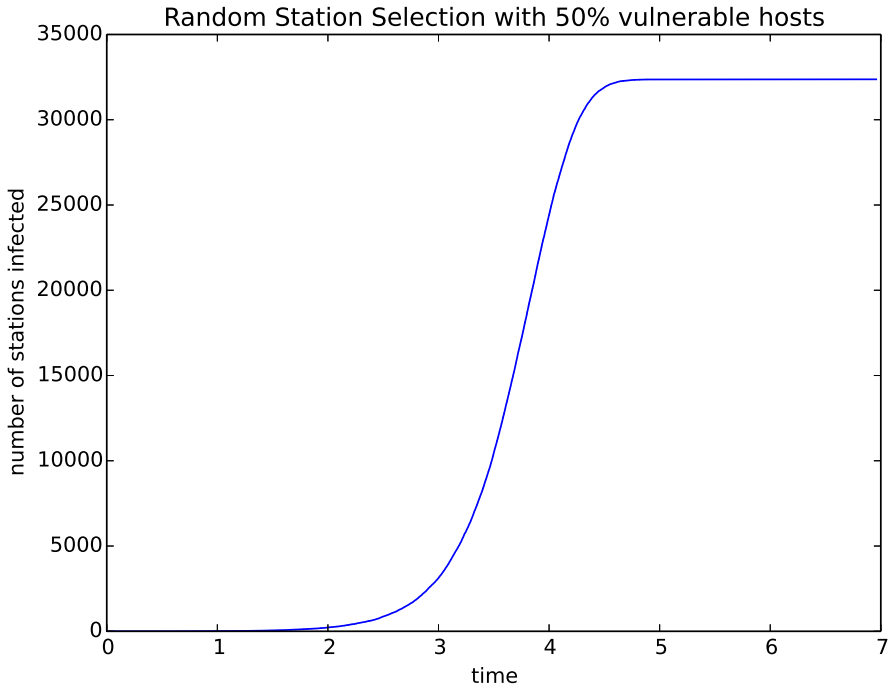

Figure 4: Random Station Selection with %50 vulnerable machines

Yes, clearly the number of infected hosts grow much faster when there are more vulnerable machines exists, this is more easy for worm to infect a machine and become a new source to propagate to more machines, the affect is exponentially growing.

2.1

In the `station.h` file, we changed the codes as follows. The random selection method is the same with last question. After many experiments, we found that the minimum value of attempts is 36, to infect all vulnerable. We tried setting the attempts to 35 first, then number of uninfected vulnerable machines ended up with 2, then I tried 40, 37 and 36, all of them successfully infect all vulnerable machines, thus 36 is the minimum value required.

```
class Station
{
public:
        Station(int id = 0):id_(id), vulnerable_(false), infected_(false),
                timeInfected_(0.0), numAttempt_(0), attemptUpper_(0)
        {  }
    ...
void propagate(double time, EventQueue& eQueue)
        {
                // find potential victim
                int toID;
                int x;
                int y;
                do{
                        x = rand()%MAX_STATIONS;          // address is 152.17.x.y
                        y = rand()%MAX_STATIONS;          // address is 152.17.x.y
                        toID = x*MAX_STATIONS + y;
                        }while( toID == id_ );  // selection excluding itself
```

```
                if(attemptUpper_ <= 36 )
                {
                        // infect attempt time
                        double infectTime = doubleUniformRV(1, 200)/1000.0;
                        eQueue.insert(EventType(Infect, time + infectTime, toID, id_));

                        // schedule next propagate attempt
                        double interPropTime = doubleUniformRV(1, 38)/1000.0;
                        eQueue.insert(EventType(Propagate, time + interPropTime, id_));

                attemptUpper_ ++;
                }
    }
    ...
private:
    ...
        int     numAttempt_;     // number of infection attempts on this station
        int     attemptUpper_;
};
```

While in the `driver.cpp` I put one extra code to see whether the

```
bool allInfected(Station* s)
{
    ...
        cout    << "The number of vulnerable left to be infected: "
                << numVulnerable - numInfected
                << endl;
        return numInfected == numVulnerable;
}
```

2.2

From `data.dat` we have

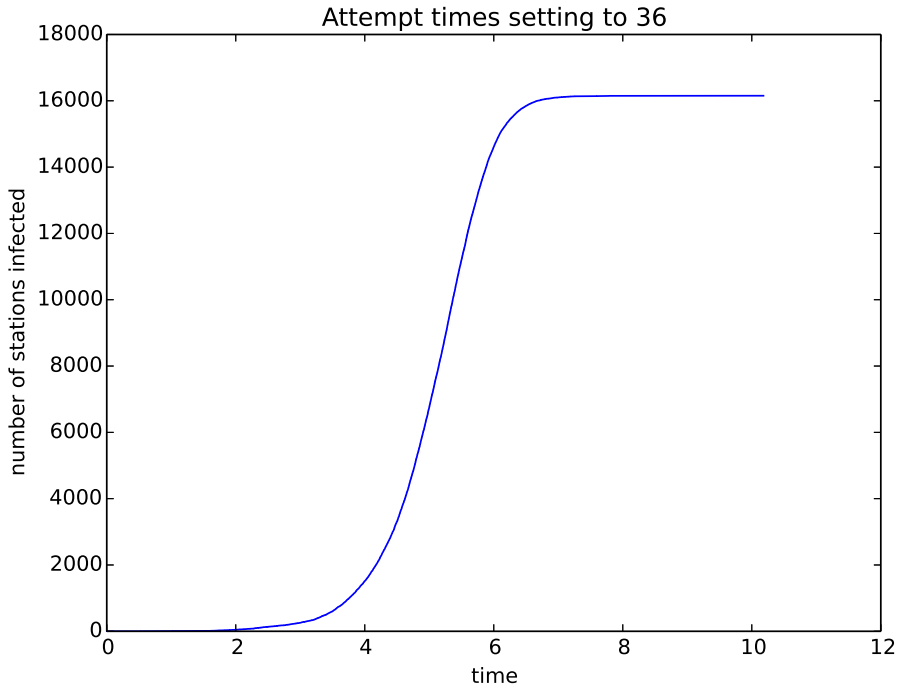

Figure 5: Number of Infected Hosts using Limited Effort Station Selection

The performance is slightly worse than random selection method. Since the number of infection attempts are upped bounded, which compromises the abilities of the worm's propagation, even though it can accomplish the mission in the end.

2.3

For `infect.dat` we have

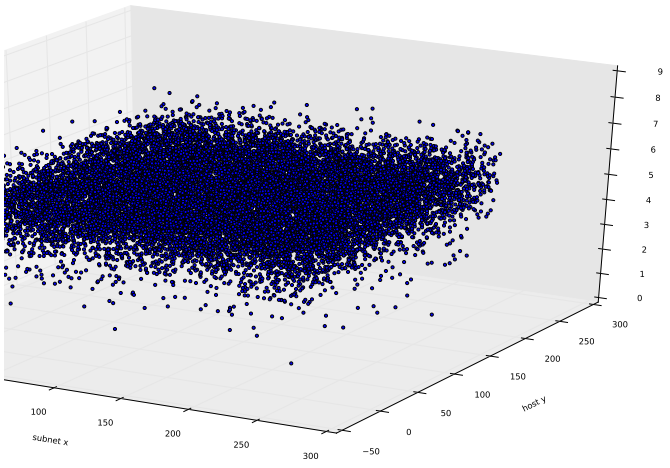

Figure 6: Initially Infected Time using Limited Effort Station Selection

No, the time is not quite the same for stations, but very close. The average is 5.0703 and the standard deviation is 0.8005 for the infect time. As the number of attempts are bounded, thus from the standard

deviation we can tell that the infect time are more centralized than random selection method, and also the average is bigger, which means slower than the previous one.

2.4

For `infect.dat` we have

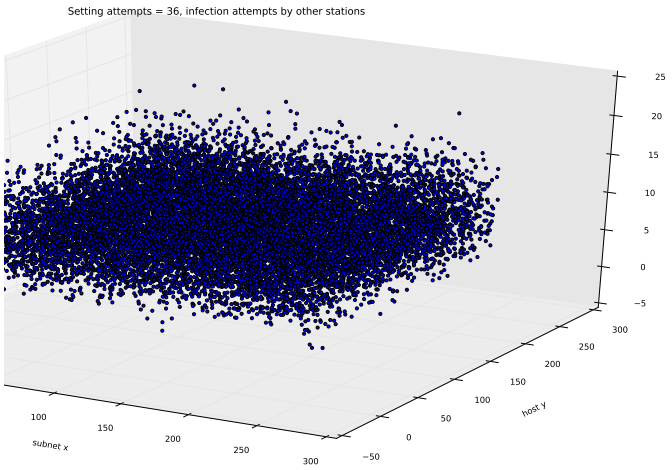

Figure 7: Attempts by other Stations using Limited Effort Station Selection

No, the attempts is not quite the same for stations, the average is 9.0984 and the standard deviation is 3.0205 for the number of the infection attempts. As the number of attempts are bounded, we can see the average the much smaller the than random selection method, whose attempts are not bounded.

3.1

In the `station.h` file, we changed the codes as follows. %75 of the time the the subnet number is the same, other than the %75 of the time we just randomly select number that is not the same with the local subnet number.

```
void propagate(double time, EventQueue& eQueue)
        {
                int toID;           // find potential victim
                int x;
                int y;
                if(double(rand())/INT_MAX <= 0.75)
                        x = row();       // select a station within its subnet
                else
                        do
                        {
                                x = rand()%MAX_STATIONS;  // address is 152.17.x.y
                        }while( x == row() );    // select from an outside subnet

                do
                {
                        y = rand()%MAX_STATIONS;          // address is 152.17.x.y
                        toID = x*MAX_STATIONS + y;
                }while( toID == id_ );                    // selection excluding itself
    ...
    }
```

3.2

From `data.dat` we have

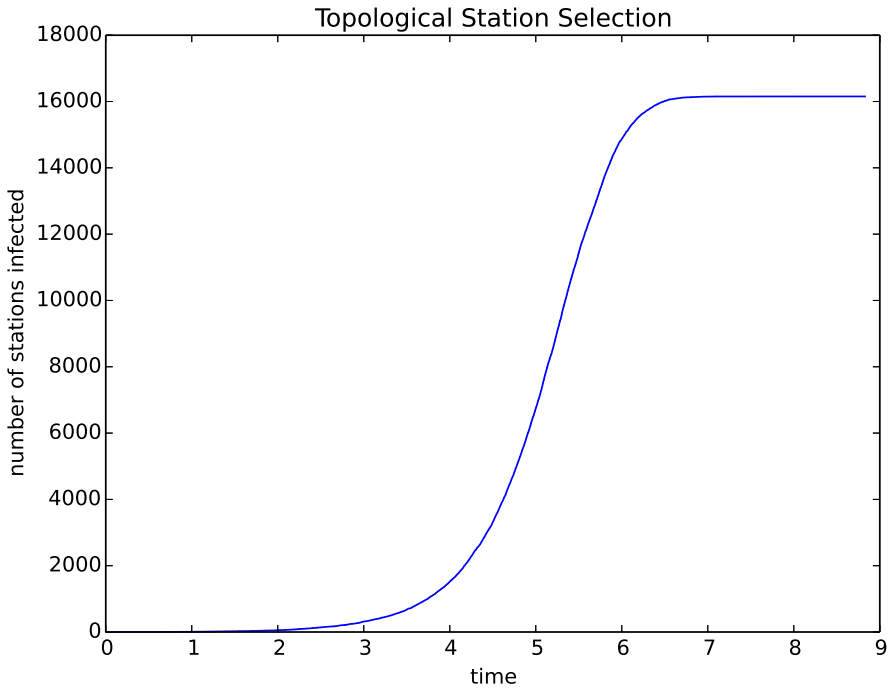

Figure 8: Number of Infected Hosts using Topological Station Selection

The performance is very close to the random selection method, just looks like only a little bit slower.

3.3

For `infect.dat` we have

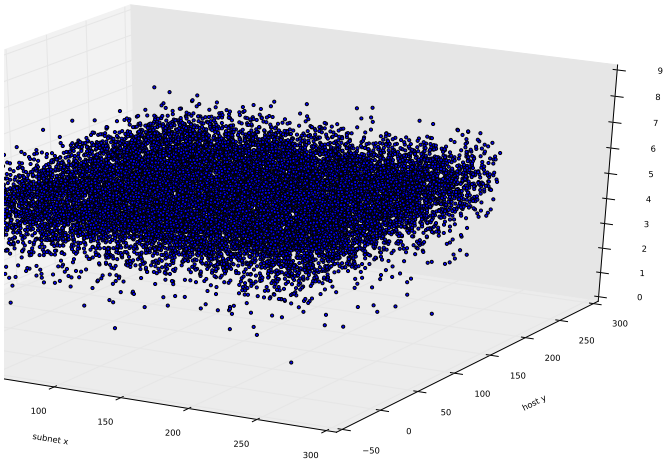

Figure 9: Initial Infected Time using Topological Station Selection method

No, the time is not quite the same for stations, but very close. The average is 5.0451 and the standard deviation is 0.7850 for the infect time. From the standard deviation we can tell that the infect time are quite

centralized than random selection method, but the average is bigger, which means slower than the previous one.

3.4

For `infect.dat` we have

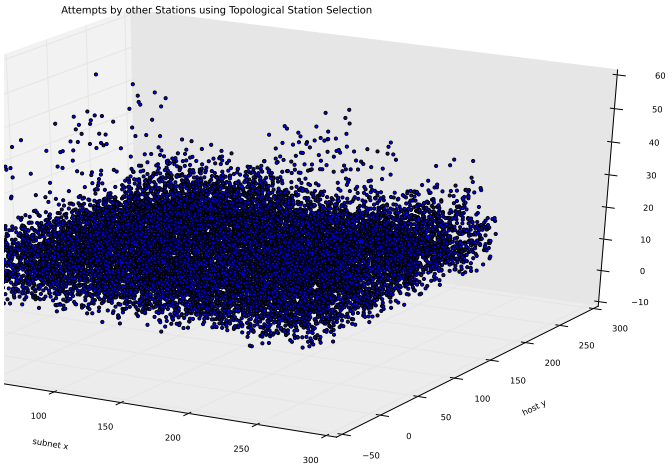

Figure 10: Attempts by other Stations using Topological Station Selection

No, the attempts is not quite the same for stations, the average is 14.5562 and the standard deviation is 6.0005 for the number of the infection attempts. This attempts are more than the two previous ones, which indicates that the performance of worm is slightly worse when the selection are more likely to be restricted to local machines.

4.1

In the `station.h` file, we assigned each station object two integers representing the beginning value and ending value of its hit list,

```
class Station
{
public:
        Station(int id = 0):id_(id), vulnerable_(false), infected_(false),
                timeInfected_(0.0), numAttempt_(0),hit_low_(1),
        hit_up_(MAX_STATIONS*MAX_STATIONS-1)
        {  }

        void setLow(int low)
        {  hit_low_ = low;  }
        void setUp(int up)
        {  hit_up_ = up;   }
        int getLow()
        {  return hit_low_;}
        int getUp()
        {  return hit_up_;}
    ...

private:
    ...
        int     hit_low_;       // begin value of the hit list
        int     hit_up_;        // end value of the hit list
```

};

We also modified the `propagate()` and `propagate()` function such that after each successful infection, their hit list will be halved and continue to propagate with the first half, while the infected host will use the second half of the hit list.

```
void propagate(double time, EventQueue& eQueue, Station* s)
{
        int toID;       // find potential victim
        if ( hit_low_ <= hit_up_ )
        {
                do
                {
                        toID = hit_low_;
                        hit_low_ ++;
                }while( id_ == toID );

                // infect attempt time
                double infectTime = doubleUniformRV(1, 200)/1000.0;
                eQueue.insert(EventType(Infect, time + infectTime, toID, id_));

                // schedule next propagate attempt
                double interPropTime = doubleUniformRV(1, 38)/1000.0;
                eQueue.insert(EventType(Propagate, time + interPropTime, id_));
        }
}

void infect(double time, EventQueue& eQueue, Station* s , int fromID)
{
        numAttempt_++; // someone attempts to infect, add to count
        if(vulnerable_ && !infected_)
        {
                if( id_ > 0)
                {
                        int fromID_low = s[fromID].getLow();
                        int fromID_up = s[fromID].getUp();
                        setLow( (fromID_low + fromID_up)/2+1);
                        setUp(fromID_up);
                        s[fromID].setUp( (fromID_low + fromID_up)/2 );
                }
                infected_ = true;
                timeInfected_ = time;
                // schedule next propagate attempt
                double interPropTime = doubleUniformRV(1, 20)/10.0;
                eQueue.insert(EventType(Propagate, time + interPropTime,id_ ));
        }
}
```

So, we also modified the `main()` function in the `driver.app` accordingly.

```
while(!eventQueue.isEmpty())
        {
    ...
                switch(currentEvent.event())
                {
                        //--Propagate-----------------------------------------------
                        case Propagate:
                                if(!allInfected(s))
                                        s[toID].propagate(time, eventQueue, s);
```

```
                        break;
                    //--Infect-----------------------------------------------------
                    case Infect:

                        s[toID].infect(time, eventQueue, s, fromID);
                        break;
                    //--illegal event----------------------------------------------
                    default:
                        cout << "Illegal event \n";
                        cout << "http://goo.gl/QMET";
                }
    ...
```

4.2

From `data.dat` we have

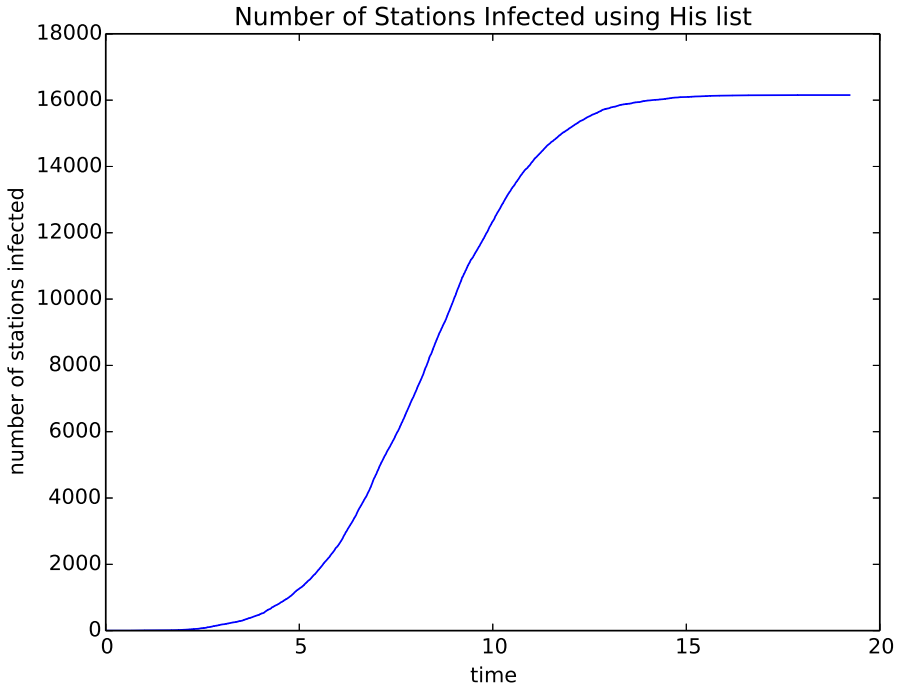

Figure 11: Number of Infected Hosts using Hit List

The performance much worse than either the random selection method or topological station selection method. Since the hit list is more like linear search, it takes longer time to find the second vulnerable machine than randomly selection, thus the burst period came later than randomly selection method.

4.3

For `infect.dat` we have

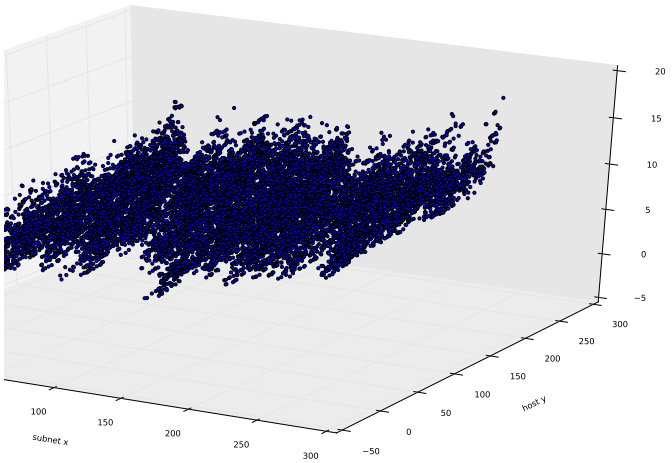

Figure 12: Initial Infected Time using His list

No, the time is not the same for stations. The average is 8.3191 and the standard deviation is 2.3487 for the infect time. As we can see here that it takes longer to infect all the vulnerable machines. Since the worm attempt to infect from smaller station ID to bigger ones on the hit list, thus the vulnerable machines with larger station ID are infected at the very end.

4.4

For `infect.dat` we have

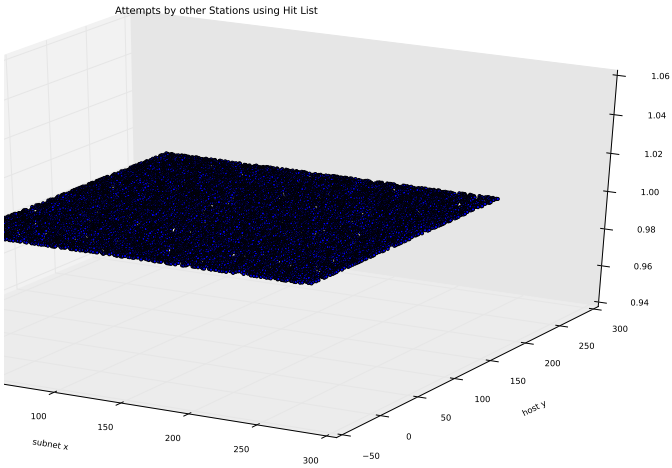

Figure 13: Attempts by other Stations using Hit List

Yes, the attempt times are exactly the same for all the stations, the average is 1 and the standard deviation is 0. This because worm is using the hit list, and will try to infect each station exactly once.

5.1

From each `data.dat` file we have above for each method, we have

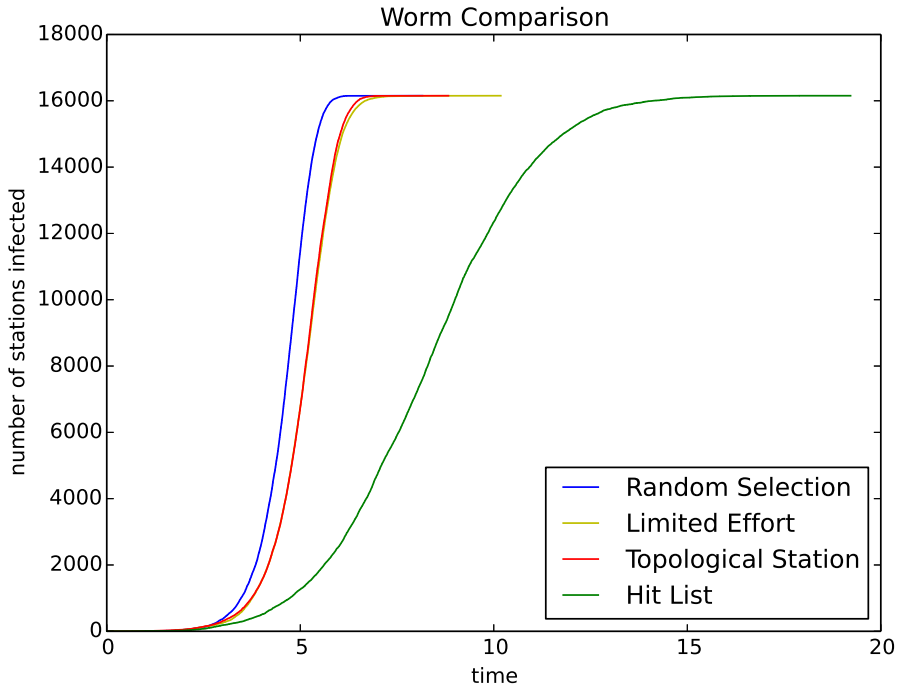

Figure 14: Worm Comparison in Number of Infected hosts over Time

As can see from the figure above, that the random selection method is the best in terms of worm propagation, Limited Effort method and Topological Station Selection method are very close to each other, and using hit list will cost the most time and infect all vulnerable machines.

6.1

Now I am using random selection as an example, to show what I changed on the codes.

In the `event.h` file, we added two more events:

```
...
enum Event{Infect, Propagate_Infect, Secure, Propagate_Secure};
...
```

In the `station.h`, we added two more functions `propagate_secure` and `secure`.

```
...
void propagate_secure(double time, EventQueue& eQueue)
        {
                // find potential victim
                int toID_s;
                int x_s;
                int y_s;
                do
                {
                        x_s = rand()%MAX_STATIONS;;  // address is 152.17.x.y
                        y_s = rand()%MAX_STATIONS;;  // address is 152.17.x.y
                        toID_s = x_s * MAX_STATIONS + y_s;
                }while( toID_s == id_ );
                // infect attempt time
                double secureTime = doubleUniformRV(1, 200)/1000.0;
                eQueue.insert(EventType(Secure, time + secureTime, toID_s, id_));
```

```
                // schedule next propagate attempt
                double interPropTime = doubleUniformRV(1, 38)/1000.0;
                eQueue.insert(EventType(Propagate_Secure, time + interPropTime, id_));
        }

void secure(double time, EventQueue& eQueue)
        {
                if(vulnerable_ && !infected_)
                {
                        vulnerable_ = false;
                        //infected_ = true;
                        timeInfected_ = time;
                        // schedule next propagate attempt
                        double interPropTime = doubleUniformRV(1, 20)/10.0;
                        eQueue.insert(EventType(Propagate_Secure,time+interPropTime,id_)
                }
        }
...
```

In the `main` function of the `station.h` file, we add those codes:

```
...
        EventQueue eventQueue;
        s[0].makeVulnerable();
        eventQueue.insert(EventType(Infect, 0.0, 0));

        s[MAX_STATIONS*MAX_STATIONS-1].makeVulnerable();
        eventQueue.insert(EventType(Secure, 0.0,MAX_STATIONS*MAX_STATIONS-1));
...

switch(currentEvent.event())
                {
                        //--Propagate-----------------------------------------------
                        case Propagate_Infect:
                                if(!allInfected(s))
                                        s[toID].propagate_infect(time, eventQueue);
                                break;
                        //--Infect--------------------------------------------------
                        case Infect:
                                s[toID].infect(time, eventQueue);
                                break;
                        case Secure:
                                s[toID].secure(time, eventQueue);
                                break;
                        case Propagate_Secure:
                                if(!allInfected(s))
                                        s[toID].propagate_secure(time, eventQueue);
                                break;
                        //--illegal event-------------------------------------------
                        default:
                                cout << "Illegal event \n";
                                cout << "http://goo.gl/QMET";
                }
...
```

We must be aware that, since the time for the bad worm to infect all vulnerable machines also includes the time the cheese worm spent on propagation and secure/patch vulnerable machine, thus, to be fair, the time for the bad worm to infect all vulnerable should be halved.

In all "fight clubs", we will always let the bad worm be contacted first, then we have to following results.

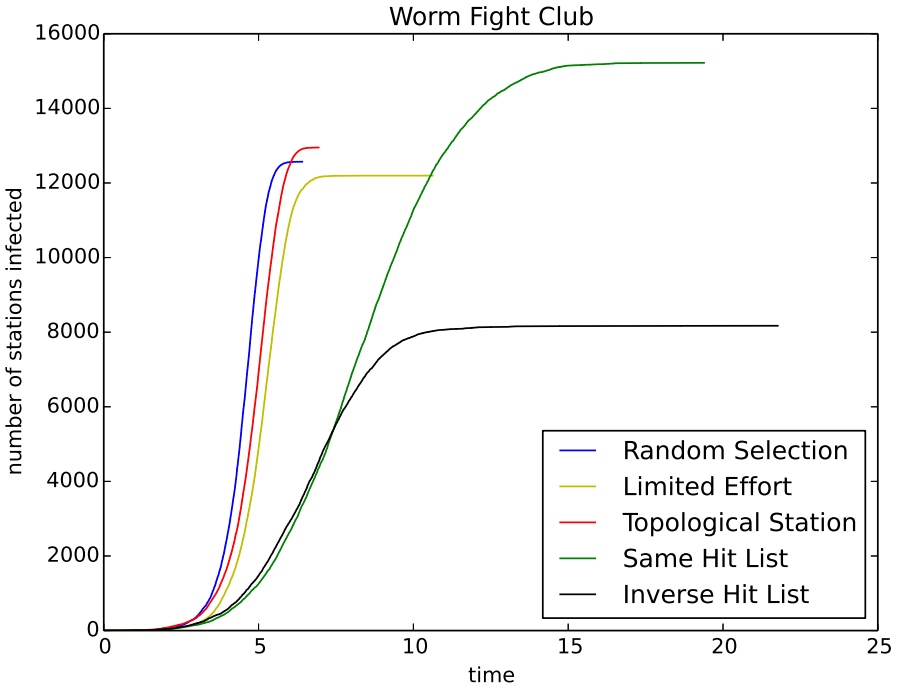

Figure 15: Worm Fight Club: Bad worms v.s. Cheese worms

As you can see we have two curves for the "hit list worms fight". For the "Same Hit List" we mean that, with a hit list that defines all hosts within the class B address range, both the good worms and bad worms are starting the first station on the list (from lower station IDs to higher station IDs). While for the "Inverse Hit List", we let the bad worms act the same way, but let the good worms start to use the last station on the hit list (from higher station IDs to lower station IDs).

Now, for the results, under the defense of cheese worm using the same selection method (except for the inverse cheese worm), the number of infected hosts are all less than the number of vulnerable machines ($254 \times 254 \times 0.25 = 16,129$), and the cheese worm using the inverse hit list only let the bad worms to achiever half of the total vulnerable machines. In terms of the time to infect all the available vulnerable machines (excluding the vulnerable hosts get patched by the cheese worms), worms using random selection method are still the fast, and the limited effort worms and topological station selection worms are not much slower than the random selection worms, while the worms using hit lists are still the slowest.

6.2

Here, our results are only based on random selection method, since this is the fastest worm that to infect all vulnerable machines, and we always let he bad worms to be contacted first. We still set the total vulnerable machines to be $254 \times 254 \times 0.25 = 16,129$.

$Experiment$	$Infected$	$Percentage(\%)$
1	$12,563$	77.89
2	$12,561$	77.87
3	$12,551$	77.81
4	$12,563$	77.89
5	$12,560$	77.87
5	$12,555$	77.84

Table 1: Experiments on Random Selection Worms

The average of infected machines are $12,560$ and quite centralized, so I don't put bar chart here. This is what I am expected, under such a big B class space, when let the bad worms be contacted first, in the end, the ratio of the infected hosts should be convergent to some fixed point.