Computer Modeling of the Spread and Containment of Computer Viruses

Adventures in Supercomputing Challenge
Final Report
April 6, 2005

Team # 054
Sandia Preparatory School

Team Members
John P. Korbin (12th)
Robbie Herbertson (9th)
Garrett Lewis (10th)
Zach Rosenberg (10th)

Teacher
Neil D. McBeth

Mentor
Erik P. Debenedictis

Table of Contents

Executive Summary

In today’s world of connectivity, computer viruses and worms pose a serious problem to America’s communication and business infrastructure. The purpose of this project was to use historical data of virus and worm attacks to create a model that will accurately simulate the activities of both viruses and worms within a given computer network. Using this model we predict vulnerabilities within a dynamic system and find solutions to alleviate these problems. The model created during this project has applications beyond our use: examples are a “Red” testing tool for system administrators that could help them predict holes in security caused by user lever changes, and a program that can use information of a network to warn a user of their personal security risks. The project’s importance is not only security of private systems but also the security of systems around the world.

We began our project with the target of designing a novel and innovative technique of modeling worms and virus. To accomplish our objective we began by learning about worms and virus, the way they are created, how they penetrate a system, and the ways that they propagate.

Educating ourselves about the world of worms and virus was a much more daunting task than we initially intended. Becoming acquainted with worms and virus led us to the domain of computer security. An area we found has a number of experts and opinions on appropriate measures to protect a particular environment. This project also facilitated our understanding of how viruses mutate, proliferate, and infiltrate a particular system.

Introduction

Significant factors in today's world of telecommunications, computer viruses and worms have the capacity, if implemented under the proper conditions, to decimate large segments of human life, from monitoring a premature infant in the hospital to launching nuclear warheads from Trident Submarines in quantities capable of destroying the world several times over. Should either one of these fail, life and capital would be lost. For that purpose, we are attempting to model a virus with given growth and detection parameters in such a manner so as to be able to better understand and more effectively fight the spread of viral infections within a system of computers.

Malicious programs (MP's) fall into two basic types those that require a host program and those that are independent. When considering MP's that require a host program a number may be involved including trap doors, logic bombs, Trojan horses, and viruses. When considering MP's that are independent, two fundamental types are considered viruses and worms.

One type of host MP, a trap door, requires a secret entry point to a program or system and can generally penetrate without the usual security access procedures. Generally, a trap door is initiated when it recognizes some special sequence of inputs, or special user identification.

Another type of host MP, a logic bomb requires an embedded entry point to a program or system and can generally explode or penetrate the legitimate program when unique pre-established conditions are met.

Another type of host MP, a Trojan horse requires a stealthy placement or hidden location in an apparently useful host program. The attack occurs or performing some unwanted/harmful function when the host program is executed.

Although these host MP's are fascinating in their own right this project focused on worms and viruses. A worm uses network connections to spread from system to system. Viruses act quite differently they "Infect" a program by modifying it and can be self-copied into the program to spread. The distinguishing factor of viruses is their four-stage life cycle. In the generalist of terms they have a dormant phase, a propagation phase (generally allowing them to be an attachment to email), a triggering phase, and an execution phase. Because of these unique qualities we chose to select these two independent MP's for our project.

To model a virus first one must understand its configuration. First there must be an identifier of the virus program, followed by a special mark to infect or not infect. The purpose of the virus program is to find uninfected programs infect and identify as vulnerable, damage or create havoc to the underlying code, and ultimately to return to programs original intent masking the fact that something is about to happen. This avoidance of detection is frequently accomplished by compressing /decompressing the original program.

Viruses themselves fall into a number of categories: parasitic viruses which search and infect executable files; memory-resident viruses that infect running programs; Boot sector viruses that spread whenever the system is booted; Polymorphic viruses which encrypt part of the virus program using randomly generated key; and Macro viruses that are actually executables embedded in a word processing document.

Anti-virus schemes therefore must consider prevention or limited contact to outside world; detection and identification; removal and various software using
simple scanners or known “signatures” of viruses; heuristic scanners using
integrity checking; activity traps that search for specific actions, and full-featured protection where all known and suspect actions are investigated.

We therefore structured our model to predict propagation and damage; understand spreading characteristics and assessed effective mitigation techniques.

To follow the spread of such a virus, one must first understand what a virus is. Essentially, it is a self-propagating code that invades a system, acting in much the same fashion as a biological virus would. Once in a system, a virus may do significant harm to a computer (Although this is not always the case) and/or quickly spread across the server before it is detected and destroyed, wreaking havoc and possible destroying several interconnected systems in the process. This is where a model applies, allowing the researcher to study several parameters at once or in quick succession without resorting to tests on closed testbed networks.

The spread of the virus then depends upon its design. It may propagate by any one of several methods including email, Internet, or individual diskettes, largely defining the rate of expansion of the virus and thus severity of same. All of these examples however, have in common the fact that they are not entirely homogeneous e-mail is affected by whether the infected message is opened, Internet by frequency of use, and diskettes by personal interaction of the people using the infected disk. Herein lies a problem with many models; they tend to treat every computer in their model as one that is perfectly equal to all others while not considering the varying infection parameters.

Also contrary to the homogeneous model is the level of protection within one system or individual. Besides the apparent differences in such consideration such as firewall protection, one must consider detection and neutralization rates within a unit and, once this process has been completed, whether the protection becomes available to all system and, if it does, at what rate. One then must consider whether a computer may be infected multiple times with a virus or that the virus may have the capacity to update itself. All of the above strictures and many more play a role in the intensity of a viral outbreak.

Description of Project

A basic program based on earlier models has been written in C++ to model the spread and detection of a virus using a system of variable parameters. Each subject computer was given beginning class of susceptible, infected, or detected. The probability of any given computer entering one of these classes is directly relative to the category before in the form of a very basic Markov chain which was moving from susceptible to infected, and then to detected over a given period of time.

In 2002, the number of known computer viruses surpassed 70,000 (www.securitystats.com). Since computer viruses work in the wild and few, if any, companies would be willing to turn off their anti-virus software to be part of a control group; their study must be done by computer modeling. One of the most important parts of the detection and analysis of viruses is the way that they spread from computer to computer. If an accurate model of the spread of viruses can be created that would keep track of the amount of the time it takes for an anti-virus company, like Norton, to create a definition of the virus and distribute it to all of its uses, it would be easier to create a temporary stopgap measure that may help keep viruses from infiltrating a network, or perhaps to find a way of identifying new kinds of viruses based on their pattern of distribution. The word virus is used to denote any malicious code. These codes fall into three main categories: virus (in technical sense), Trojan horses, and worms.

Using C++ it was possible to create a program, building on an already written program by HP, that created an on the fly model of how the viruses propagate which allowed manipulation of the time needed to create and distribute an anti-virus definition. The model was based on the following premise of how the average virus spreads:

1. The virus is released into the wild by its creator
2. The virus spreads freely, infecting machines and delivering its payload
3. The virus is eventually noticed and the company is alerted
4. The company then works to isolate the virus and generate a "signature" that can be used in scanning software to detect the presence of the virus
5. Then the company distributed the signature to its clients through a central server regularly polled by the client machines looking for updates

The goal was to use a computer or cluster node to create a model in graphical form of how the virus spreads initially and then to monitor how long it took to purge all, or most of, the computers of the virus. Note: no "live" viruses were used in this model. The machines (in the model) moved between four states:

1. Susceptible machines which are vulnerable to infection. (S)
2. Infected machines that have contracted the virus and are actively spreading it. (I)
3. Machines in which the viruses has been detected and are prevented from spreading the virus. The state includes computers that have been disconnected from its network when a virus is

detected to prevent further spreading of the virus as well as computers that have been incapacitated by the virus and thus, are inactive and no longer spreading the virus. (D)

The removed state consists of machines that are immune to the virus. The state includes machines that are not susceptible to the virus to begin with as well as machines in which the virus has been removed and is made immune to the virus by the anti-virus software

Markov Chains

The mathematics behind the project is centered on the Markov Chain. Named for the Russian mathematician, Andrei Markov, it is simply defined as a series of events in which the probability of one event occurring is based solely and wholly on the preceding event. This may be exemplified by a person throwing free throws in a basketball game. If he makes his first shot, he may feel confident about himself and as such improve his chances of making the second shot. However, if he misses his first shot, he may become dejected, hurting his chances at making the second shot. Thus, we have the matrix equation:

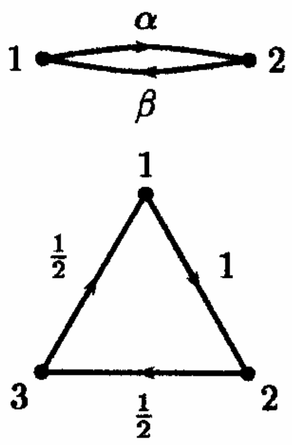
$$\left[P(B_2)\cdot P(M_2)\right] = \left[P(B_1)\cdot P(M_1)\right]\begin{bmatrix} P(B_2 \mid B_1) & P(M_2 \mid B_1) \\ P(B_2 \mid M_1) & P(M_2 \mid M_1) \end{bmatrix}$$

Where P(B) is the probability of making a basket and P(M) refers to the probability of missing the basket. The first matrix refers to the probability of the events (make/miss) occurring on the second shot, the second matrix indicating the probability of the events occurring on the first shot, and the third matrix referring to the probability of either event occurring in the second shot relative to the outcome of the first. A general matrix similar to the third above appears below with a 3x3 matrix to which numerical values have been applied. Note that the sum of the values within the latter matrix add to a total probability of 1.

[1]

$$P = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}$$

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \end{pmatrix}$$



At this point, we will replace the matrices with symbols to simplify the equation's appearance such that,

$$P_2 = P_1 \prod$$

Where each letter represents its respective matrix.

If the basketball player were to continue shooting while reacting to his successes and failures in the same manner as above, the equation, it would stand to reason, would simply continue to expand on each successive probability by multiplying it by $\prod$ as follows:

$$P_3 = P_2 \prod$$

$$= (P_1 \prod)^* \prod$$

$$= P_1 \prod{}^2$$

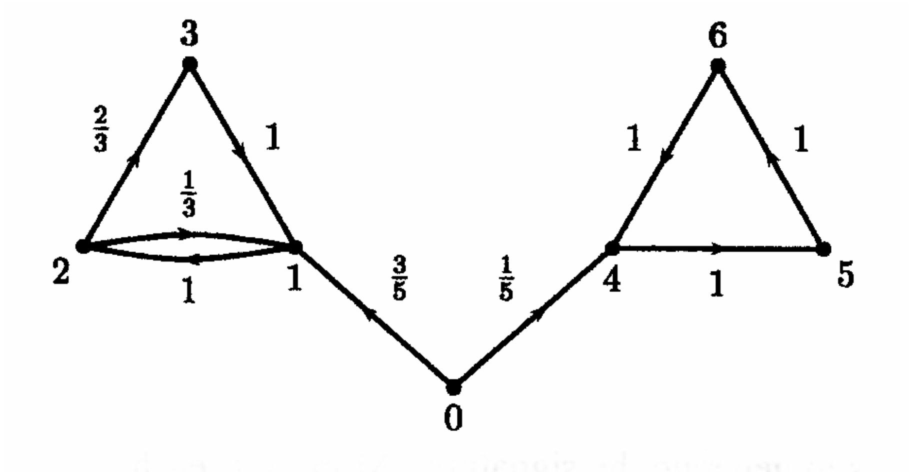
Therefore, we see that:

$$P_n = P_1 \prod{}^{n-1}$$

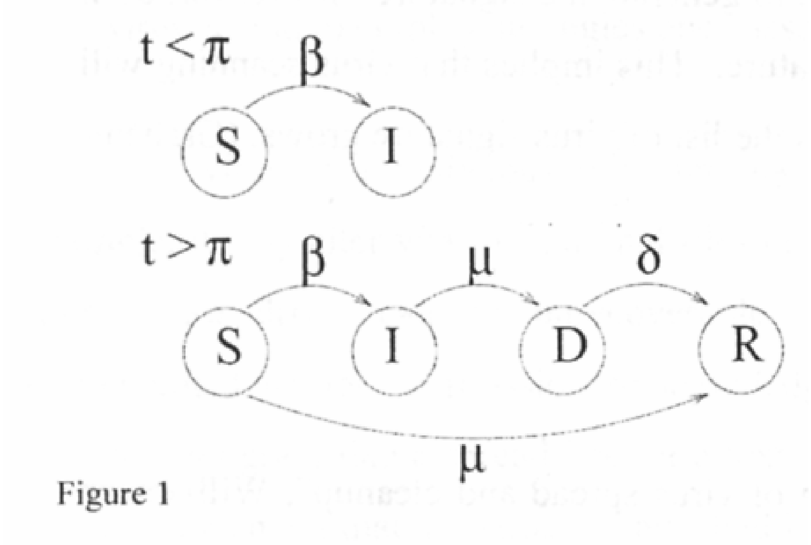
This equation gives the probability of making (or missing) the nth shot in any series as a part of a continuous Markov Chain. It is to be noted, however, that at any given point in the series, the probability of making the immediately following shot is drawn from $\prod$ with only the data from the shot immediately before, not from the entire equation. It must also be recognized that as a system expands to contain states, as in the case of this project, that $\prod$ may expand to become a significantly larger matrix.

Up until this point every example has been one in continuous time, that is, the nodes continuously rotate among the various states. However, in virus research the ultimate goal is the elimination of the virus by immunizing all of the vulnerable nodes, and as such, a continuous chain in which the computers return to a vulnerable state becomes undesirable. Here we find a use for discrete time in which not all nodes communicate with each other and in which many only have one way communication, as in the case of this study and computer virus research in general. Two examples of such a chain may be seen below, the latter being quite similar to the model used herein.

2



3



Here, we find four states- susceptible, infected, detected, and removed- arranged in such a manner so as to move only forward, that is, from left to right on this diagram. This motion prevents the virus from returning to the system once it is removed, and thus represents the desired model for a good model of a virus protection program.

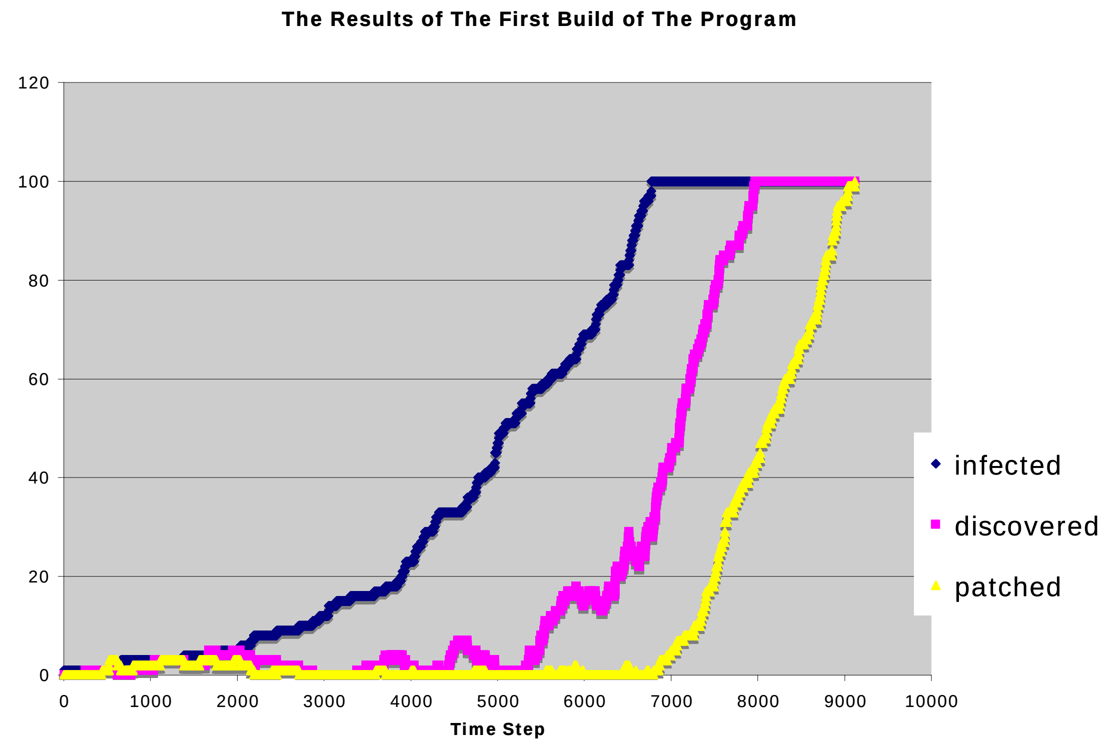
Results

The results of the first build of our program (Appendix A) demonstrated the average virus cycle. At first a single computer is infected, and then the virus begins to spread to other computers. Eventually people and/or anti-virus software identifies that a computer is infected. Lastly the computer is patched, eliminating the virus and removing the weakness in the computer software that the virus exploited to infect the computer.

The goal for this project was to simply attempt to recreate, as accurately as possible, of an average virus attack. Usually a virus's infection rate increases exponentially as the number of infected computers increases; our results matched this pattern (see the graph bellow).

Because of the rather limited computer power available, this model has been simplified to make it easier to look at just the initial growth rate of the virus and does not take into account the possibility that a computer could be re-infected or that an infected computer could be in any number of infected states; if a computer is infected it is actively spreading the virus.

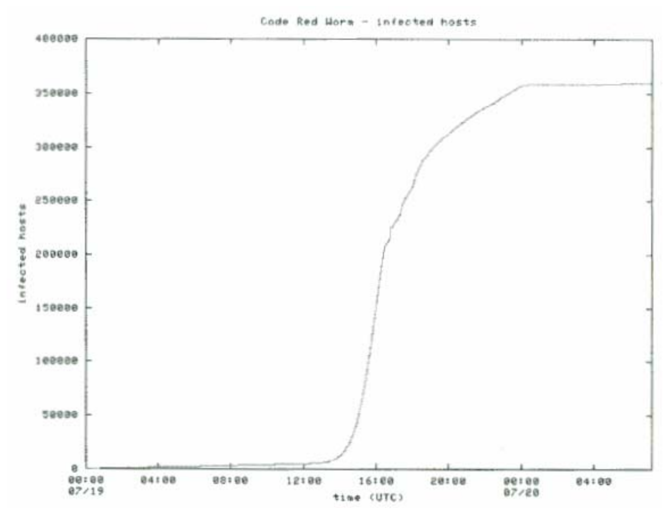
This particular virus took ten thousand time steps, a time step being any set amount of time: second, minute, hour, day, etc., to infect the entire population of 100 computers and for them to be identified as infected and subsequently patched.



We learned that, while being able to predict how a virus will spread is a very important endeavor, it is very hard to create a model that takes into account every variable, especially the human ones. While we were successful at creating a computer program that accurately simulates a virus attack, it is still limited in its abilities.

Conclusions

Overall, the project goals were attained as the project successfully modeled the spread of a hierarchically propagating computer virus through a system of nodes while allowing the user to define the spread parameters as he or she sees fit. This design allows the model to be used to study the propagation of a virus, real or hypothetical, with any desired parameters, thus allowing for an improved understanding of its ability, or lack thereof, to damage a system and possibly leading to an improved opportunity to implement security measures.



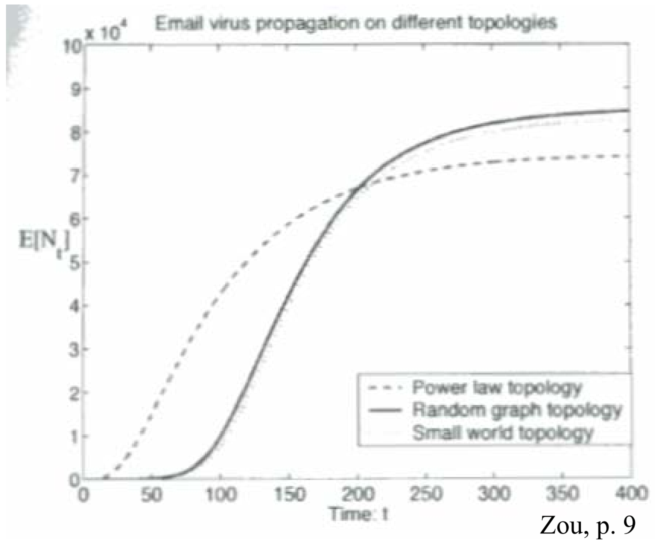
The above diagram shows the rapid expansion of Code Red v.2 from its first outbreak in 2001

Serazzi, p. 8.

The ability to modify the spread parameters in the program is especially important in modeling the superabundance of viruses in today's world where there are innumerable variations in every type of virus possible, from primitive disk viruses to Code Red, which quickly implanted itself among its target population almost immediately after its release. Code Red was, of course, a scanning virus and therefore expanded along a slightly different curve than e-mail viruses, which are better suited to this program due to their hierarchical nature.

The e-mail viruses toward which the program is directed are quite important in today's world of ever-expanding usage. Yahoo! Groups, for example, have thousands of groups ranging in size from as low as four members to well over 100,000 members,[4] where each of the e-mail groups serves the role of a group of nodes hierarchically connected to each other. Should a virus strike such a large population of nodes and successfully propagate across the system, there could be untold economic damage, and it is to the prevention of this end that this project is geared, a goal which has been, to an extent, achieved in the model by allowing researchers to predict the virus's expansion before it is released and prepare for such an event. This comparison may be seen between the program's data and that which is pictured in the diagram below.

4 Zou, "E-mail Virus," p. 5.



Zou, p. 9

Although the program does not allow for discrepancies and variations in individual nodes, such as connectivity, bandwidth, and anti-virus software, one sees that the general trend is accurate and trustworthy, varying only in the rate of propagation as defined by the user. This alone in itself may not offer a solution for a viral attack, but it gives researchers data pertaining to the time in which they have to react and the time required to disinfect the system once the process has been initialized. This greatly aids in preventing overreacting as well as excessive leniency should an outbreak occur, thus improving the overall actions taken against the intrusion.

Recommendations

The program could be easily modified to create a more accurate model in a couple of ways. First, modify the program to allow more variables on each individual node. The problem with this kind of program is the fact that it requires a lot of computer power and is very slow on the laptop we were using to run the program. The program is also designed to allow the user to modify the number of clients in the model; however, any more than about hundred nodes would be very slow and hard to run on the computers to which we had access.

Our most significant achievement on the project can clearly be stated in that we learned a lot about mathematics and computer virus spread. We also learned that creating a model of a large-scale computer network requires a lot of computing power to be both accurate as well as executable.

Acknowledgements

Our team would like to thank the Sandia Preparatory School administration and staff for providing us with the opportunity and financial support to participate in the challenge. In particular, we acknowledge the transportation, school computer resources, after-school meetings, and overall enthusiasm provided by SPS. We appreciate our advisor Mr. Neil D. McBeth for the encouragement, tenacity, and patience to endure our difficulties.

We wish to express our sincere thanks to our Sandia National laboratory Technical advisor, Mr. Erik P. Debenedictis. His real world experience with the Red Storm Project brought to the project a unique perspective. Further, his drive, creativity, and confidence in our abilities truly mentored and inspired each of the team members.

We wish to express our thanks to Mr. Edward Bedrick of the University of New Mexico Statistics department. His knowledge and application of statistics greatly influenced our project.

We appreciate the support and affirmation provided by our Parents

Finally, we wish to thank all those that sponsor the supercomputing challenge for the opportunity.

Bibliography

Billings, Lora, William M. Spears and Ira B. Schwartz. "A unified prediction of computer virus spread in connected networks." Physics Letters.

Chen, Zesheng, Lixin Gao and Kevin Kwiat. "Modeling the Spread of Active Worms."

Chess, David M. and Steve R. White. "An Undetectable Computer Virus."

Eugene, Ng Aik Meng, Chan Ying Ming Colin and Choo Yanqing. "An Epidemiological Model of Computer Virus Spread and Countermeasures." http://staff.science.nus.edu.sg/~parwani/sim/simproject2/computervirus.doc.

Kephart, Jeffery O. and Steve R. White, "Directed-Graph Epidemiological Models of Computer Viruses." ANTIVIRUS RESEARCH - Scientific Papers.

http://www.research.ibm.com/antivirus/SciPapers/Kephart/VIRIEEE/virieee.gopher.html (5 Jan 2005).

Moore, David, Colleen Shannon, Geoffrey M. Voelker and Stefan Sabage. "Internet Quarantine: Requirements for Containing Self-Propagating Code."

Nachenberg, Carey. "Understanding and Managing Polymorphic Viruses." http://www.madchat.org/vxdevl/vdat/epunders.htm (14 Feb 2005).

Nazario, Jose, Jeremy Anderson, Rick Wash and Chris Connelly. "The Future of Internet Worms." http://www.crimelabs.net.

Norris, James, "Markov Chains," http://www.statslab.cam.ac.uk/~james/Markov/. Cambridge University Press, 2004.

Serazzi, Giuseppe and Stefano Zanero. "Computer Virus Propagation Models."

Vogt, Tom. "Simulating and optimizing worm propagation algorithms."

Wang, Chenxi , John C. Knight and Matthew C. Elder. "On Computer Viral Infection and the Effect of Immunization."

White, Steve R. "Open Problems in Computer Virus Research." http://www.research.ibm.com/antivirus/SciPapers/White/Problems/Problems.html (14/feb/2005).

Williamson, Matthew M. and Jasmin Leveille. "An epidemiological model of virus spread and cleanup." http://www.hpl.hp.com/techreports/2003/HPL-2003-39.pdf.

Zou, Cliff C., Don Towsley and Weibo Gong. "Email Virus Propagation Modeling and Analysis."

Zou, Cliff Changchun, Lixin Gao, Weibo Gong and Don Towsley. "Monitoring and Early Warning for Internet Worms." CCS'03, October 27-30, 2003, Washington DC.

Zou, Cliff Changchun, Weibo Gong and Don Towsley. "Worm Propagation Modeling and Analysis under Dynamic Quarantine Defense." Worm'03, October 27, 2003, Washington DC.

Appendix A Code Build 1

```
// HamEggAndSpam.cpp : Defines the entry point for the console application.
// Better random numeber gen
#include <stdafx.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>
#include <string.h>
#include <time.h>

ofstream fout;

struct computer{

 bool infected;
 computer * friends [10];

};

int main(void)
{
 float infection = 0;
 float patch = 0;
 float discovered = 0;
 long num_computers = 0;
 long counter = 0; //counter for first for loop
 long infected_computers = 1;
 long discovered_computers = 0;
 long patched_computers = 0;
 int time = 1; //starting time step
 double time_discovery;
 double time_patch;
 int a;
 char sys_time[15];
 char sys_date[15];

 _strtime(sys_time);
 _strdate(sys_date);


#if 0

 do
 {

  cout <<"Enter Infection constant: ";
  cin >>infection;

 }while (infection < 0 || infection > 1);
 do
 {

  cout <<"Enter Discovery constant: ";
  cin >>discovered;

 }while (discovered < 0 || discovered > 1);

 do
 {

  cout <<"Enter Patch constant: ";
  cin >>patch;

 }while (patch < 0 || patch > 1);

 do
 {

  cout <<"Enter Number of Computers: ";
  cin >>num_computers;

 }while (num_computers <= 0);



 do
 {

  cout <<"Enter time_discovery: ";
  cin >>time_discovery;

 }while (time_discovery <= 0);


 do
 {

  cout <<"Enter time_patch: ";
  cin >>time_patch;

 }while (time_patch < time_discovery);

#else

 infection = 0.5; //chance of getting infected
 discovered = 0.5; //chance of getting discovered to be patched
 patch = 0.5; //chance of getting patched 0.5 = 50:50
 num_computers = 100; //number of computers in the network
 time_discovery = 100; //time it takes to get discovered (in steps)
 time_patch = 100; //time untill a patch is made (in steps)

 fout.open("output.txt", ios::app); //outputfile open
```

```
#endif

 double x;
 double y;
 double b;
 double c;
 double d;
 double e;
 cout << "0, " << infected_computers << "\n";
 computer * net = new computer[num_computers];

 for (int i = 0; i < num_computers; i++){

  net[i].infected = 0;

  for (int j = 0; j < 10; j++){

   net[i].friends[j] = 0;

  }

  net[i].friends[0] = i==0 ? 0 : &net[i-1];
  net[i].friends[1] = i==9 ? 0 : &net[i+1];


 }

/* do
 {
   counter = 1;
   while (counter <= (num_computers - infected_computers))
   {

    x = rand()/(double)RAND_MAX;
    y = (infection*((double)infected_computers/(double)num_computers)/1000);   // 2 is a constant that has no real
reason to be there

    if ( x <= y )
    {
     infected_computers++;
    }

    counter++; //increments counter by one

   }

   cout << time << ", " << infected_computers <<"\n";

   time++;


 }while (time < time_discovery);
 cout<<"discover" << "\n";

 //there is a problem in the logic below that needs to be corrected
 */

 do
 {

  counter = 1;

  while (counter <= num_computers)
  {
   x = rand()/(double)RAND_MAX;
   y = (infection*((double)infected_computers/(double)num_computers)/1000);

   if (patched_computers < num_computers && x < y && infected_computers < num_computers &&
infected_computers < (num_computers-patched_computers))
   {
    infected_computers++;

    if (discovered_computers > 0)
    {
     discovered_computers--;
    }
    if (patched_computers > 0)
    {
     patched_computers--;
    }
   }

   b = rand()/(double)RAND_MAX;
   c = (discovered*((double)infected_computers/(double)num_computers)/1000);

   if (infected_computers >= 1 && b < c && discovered_computers < num_computers)
   {
    discovered_computers++;
    //infected_computers--; <- major fix to logic
   }

   d = rand ()/(double)RAND_MAX;
   e = (patch*((double)discovered_computers/(double)num_computers)/1000); //<- /1000 was *1000

   if (discovered_computers >= 1 && d < e && patched_computers < num_computers)
   {
    patched_computers++;
   }

    if (infected_computers > patched_computers && infected_computers + patched_computers == 100 &&
patched_computers > 0)
    {
     infected_computers--;
    }
    if (infected_computers < patched_computers && infected_computers + patched_computers == 100)
    {
     patched_computers--;
    }
   counter++;


  }

  cout << time << ", " << infected_computers << ", " << discovered_computers<< ", " << patched_computers<<
"\n";
#if 0

  fout << time << ", " << infected_computers << ", " << discovered_computers<< ", " << patched_computers<<
"\n"; //outputfile write

#endif
  time++;

 }while (patched_computers < num_computers);

 fout <<"----------------------------------------" <<"\n";
 fout <<sys_time <<" "<<sys_date <<"\n";
 fout <<"----------------------------------------" <<"\n";

 delete net;
 cout <<"Done. Press any key to exit/> ";
 cin >>a;
 return 0;
}
```

Code Build 2

```
// virusim.cpp : Defines the entry point for the console application.
// Better random numeber gen needed <--- DONE being tweaked!!
#include <stdafx.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>
#include <string.h>
#include <time.h>
#include "randomc.h"
#include "mersenne.cpp"
#include "userintf.cpp"

 int32 seed0 = time(0);
 TRandomMersenne rg0(seed0);

 int32 seed = time(0);
 TRandomMersenne rg(seed);

 int32 seed1 = time(0);
 TRandomMersenne rg1(seed1);
 int32 seed2 = time(0);
 TRandomMersenne rg2(seed2);

 float infection = 0;
 float patch = 0;
 float discovered = 0;
 long num_computers = 0;
 long counter = 0; //counter for first for loop
 long infected_computers = 1;
 long discovered_computers = 0;
 long patched_computers = 0;
 int time1 = 1; //starting time step, also changed from name to time due to new RNG implementation
 double time_discovery;
 double time_patch;
 int a;
 char sys_time[15];
 char sys_date[15];

 double r;                    // random number
 int32 ir;




ofstream fout;

struct computer
{
 bool infected;
 bool patched;
 bool discover;
 computer * friends [10];

};

int main(void)
{
 _strtime(sys_time);
 _strdate(sys_date);


#if 0

 do
 {

  cout <<"Enter Infection constant: ";
  cin >>infection;

 }while (infection < 0 || infection > 1);

 do
 {
  cout <<"Enter Discovery constant: ";
  cin >>discovered;

 }while (discovered < 0 || discovered > 1);

 do
 {

  cout <<"Enter Patch constant: ";
  cin >>patch;

 }while (patch < 0 || patch > 1);

 do
 {

  cout <<"Enter Number of Computers: ";
  cin >>num_computers;

 }while (num_computers <= 0);



 do
 {

  cout <<"Enter time_discovery: ";
  cin >>time_discovery;

 }while (time_discovery <= 0);


 do
 {

  cout <<"Enter time_patch: ";
  cin >>time_patch;

 }while (time_patch < time_discovery);

#else

 infection = 0.5; //chance of getting infected
 discovered = 0.5; //chance of getting discovered to be patched
 patch = 0.5; //chance of getting patched 0.5 = 50:50
 num_computers = 100; //number of computers in the network
 time_discovery = 100; //time it takes to get discovered (in steps)
 time_patch = 100; //time untill a patch is made (in steps)

// fout.open("output.txt", ios::app); //outputfile open

#endif

/* double x;
 double y;
 double b;
 double c;
 double d;
 double e;
*/

 cout << "0, " << infected_computers << "\n";
 computer * net = new computer[num_computers];
/*
 for (int i = 0; i < num_computers; i++){

  net[i].infected = 0;

  for (int j = 0; j < 10; j++){

   net[i].friends[j] = 0;

  }

  net[i].friends[0] = i==0 ? 0 : &net[i-1];
  net[i].friends[1] = i==9 ? 0 : &net[i+1];


 }

 do
 {

  counter = 1;

  while (counter <= num_computers)
  {
   x = rg.Random();///(double)RAND_MAX;
   y = (infection*((double)infected_computers/(double)num_computers)/1000);

   if (patched_computers < num_computers && x < y && infected_computers < num_computers &&
infected_computers < (num_computers-patched_computers))
   {
    infected_computers++;

    if (discovered_computers > 0)
    {
     discovered_computers--;
    }
    if (patched_computers > 0)
    {
     patched_computers--;
    }
   }

   b = rg1.Random();
   c = (discovered*((double)infected_computers/(double)num_computers)/1000);

   if (infected_computers >= 1 && b < c && discovered_computers < num_computers)
   {
    discovered_computers++;
    //infected_computers--; <- major fix to logic
   }

   d = rg2.Random();
   e = (patch*((double)discovered_computers/(double)num_computers)/1000); //<- /1000 was *1000

   if (discovered_computers >= 1 && d < e && patched_computers < num_computers)
   {
    patched_computers++;
   }

    if (infected_computers > patched_computers && infected_computers + patched_computers == 100 &&
patched_computers > 0)
    {
     infected_computers--;
    }
    if (infected_computers < patched_computers && infected_computers + patched_computers == 100)
    {
     patched_computers--;
    }

   counter++;


  }

  cout << time1 << ", " << infected_computers << ", " << discovered_computers<< ", " << patched_computers<<
"\n";
#if 1

  fout << time1 << ", " << infected_computers << ", " << discovered_computers<< ", " << patched_computers<<
"\n"; //outputfile write

#endif
  time1++;

 }while (patched_computers < num_computers);


 fout <<"----------------------------------------" <<"\n";
 fout <<sys_time <<" "<<sys_date <<"\n";
 fout <<"----------------------------------------" <<"\n";


 delete net;
 cout <<"Done. Press any key to exit/> ";
 cin>>a;
 */

 return 0;
}

get_random_friends (int i, computer*net) //the number of friends can not excede 10
{
 double number_friends = rg0.IRandom(0,10);
 for (int q = 0; q < number_friends; q++)
 {
  if (net[i].friends[q] == 0)
  {
   net[i].friends[q] = &net[int((rg0.Random()*num_computers))];
   for (int j = 0; j < 10; j++)
   {
    if (net[q].friends[j] == 0)
    {
     net[q].friends[j] = &net[i];
    }
   }
  }
 }
  return 0;
}

infect_net(int i, computer*net) //new way of getting infected
{
 double x = rg.Random();
 double y = (infection*((double)infected_computers/(double)num_computers));

 for ( int z = 0; z < 10; z++)
 {
  if (x < y && net[i].friends[z]->infected == 1 && net[i].infected == 0 && net[i].patched == 0)
  {
   net[i].infected = 1;
   infected_computers++;
  }
 }
 return 0;
}

discover_net (int i, computer*net) //new way of discovery
{
 double x = rg1.Random();
 double y = (infection*((double)infected_computers/(double)num_computers));

 if (net[i].infected == 1 && net[i].discover == 0 && x < y)
 {
  net[i].discover = 1;
  discovered_computers++;
 }
 return 0;
}

patch_net (int i, computer*net) //new way of getting patched
{
 double x = rg2.Random();
 double y = (infection*((double)infected_computers/(double)num_computers));

 if (net[i].discover == 1 && net[i].patched == 0 && x < (y*100)) // it is now 100 times more likely to get a patch if
it is infected
 {
  net[i].patched = 1;
  infected_computers--;
 }
 else if (net[i].discover == 0 && net [i].patched == 0 && x < y)
 {
  net[i].patched = 1;
  infected_computers--;
 }
 return 0;
}
```